QUESTION-ANSWER INPUTS TO A SIMULATION PROGRAM GENERATING SYSTEM

Eldon Stover Baker

United States
Naval Postgraduate School

THESIS

QUESTION-ANSWER INPUTS
TO
A SIMULATION PROGRAM GENERATING SYSTEM

by

Eldon Stover Baker

Thesis Advisor: G.E. Heidorn

June 1971

Approved for public release; distribution unlimited.

Question-Answer Inputs to A Simulation Program Generating System

by

Eldon Stover Baker
Lieutenant Commander, United States Navy
B.S., Manchester College, 1957

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1971

ABSTRACT

One of the objectives of a research effort at the Naval Postgraduate School is the realization of a "simulation program generating system" to produce executable computer programs from natural language descriptions of simulation problems. The basic part of the system being developed is a FORTRAN program for translating input text into a representative data structure and constructing output text from the data structure. Previous work has resulted in the capability to translate the data structure for a simulation problem into an English text description and a GPSS computer program. The purpose of this thesis was to supplement the above with an interactive question-answer scheme that generates the simulation problem data structure. The resulting GPSS "simulation program generating system" is capable of handling a variety of simple queuing problems.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

System analysis through computer simulation is widely recognized as a valuable tool for optimizing system performance and reducing operating cost. The process of creating a usable computer simulation model ultimately results in the need to transform a conceptual model of the system being studied into a suitable computer program. Creating this program requires specialized knowledge in computer simulation techniques and programming. A system which could automate this task would significantly reduce the time and cost of producing a computerized model.

A. PROGRAM GENERATING SYSTEMS

Higher-level computer programming systems such as FORTRAN, ALGOL, or PL/1 may be loosely classified as "program generating systems." In this context, the programmer writes statements that comply with the syntax and semantics of the language. A compiler then generates an executable machine language program from the statements provided.

In this thesis, however, "program generating system" or "program generator" implies a method of obtaining a usable computer program (possibly in a higher-level programming language) without having to write statements in a programming language. In a system of this type, the problem to be programmed might be stated by filling out a questionnaire, where the answers could be key-punched on computer cards,

as input data for the program generator, which would then produce the desired computer program. Reference 1 describes such a system developed at The RAND Corporation. The RAND system employs decision tables and a statement list of computer commands in the output language. The decision tables specify the command or series of commands required from the statement list as a function of the choices selected on the questionnaire. Reference 2 describes in detail a specific application of this system to "job shop" simulation programming.

Another type of program generating system could be one which accepts as input data an English language statement of the problem to be programmed. This presents the problem of translating English language into a usable computer program, or into a computer data structure which adequately represents the information in the English text. This task falls into the realm of "natural language processing."

B. NATURAL LANGUAGE PROCESSING

Reference 3 reviews much of the recent work done in the field of natural language processing. It discusses, primarily, systems which (1) answer questions, (2) interactively engage the user in apparently intelligent conversation, or (3) solve word-problems. Natural language processing involves, in some form, a method of transforming or decoding the English text into a computer data structure and a system which analyzes the data structure, and encodes an

appropriate output. A "natural language program generator" would, of course, produce a computer program as the final output.

At the Naval Postgraduate School, an interactive system has been developed to perform the decoding and encoding processes mentioned above. This system, called NLP (Natural Language Processor) is capable of translating input text into a data structure by following "decoding rules," and then analyzing the data structure and producing output by following "encoding rules." NLP is presently configured to operate on the IBM 360/67-CP/CMS time-sharing system at the Naval Postgraduate School. With minor modification, however, NLP could operate on most time-sharing systems with a FORTRAN IV capability.

C. BACKGROUND

Current research with NLP is directed toward the problem of decoding a natural language description of a computer simulation problem and generating (encoding) an appropriate computer program to perform the simulation [Ref.4]. Decoding rules have been written to translate simple English sentences into a data structure of linked records. These records contain "attributes" and "indicators" which describe the content and meaning of a sentence. Corresponding encoding rules have been written to process the data structure and encode sentences according to the attributes and indicators of the records.

An appropriate internal data structure (IDS) for simple queuing problems and a set of NLP encoding rules, called GES (GPSS Encoding System), that produce computer programs in the GPSS simulation language, have also been developed, as reported by Hansen [Ref.5]. In addition, McGee [Ref.6] describes the development of XGES, an expanded and enhanced version of GES, and the development of a set of encoding rules which produce, from the IDS, a natural language statement of the simulation problem as "understood" by the computer.

D. OBJECTIVE

The objective of the research reported on in this thesis was to design and implement, within the framework of NLP, a procedure for developing simulation programs by interactive question-answering at a computer time-sharing terminal. The result of this work is a set of decoding rules and the FORTRAN subprograms needed for NLP to function as a "program generating system" for simple queuing simulation problems. This research has also provided some basic techniques and background data that can be used in the development of the full natural language processing capability.

E. ORGANIZATION OF THESIS

The remainder of the thesis is divided into four sections. Section II describes the relevant portions of NLP, in particular the decoding process. Section III covers the question-answer scheme and the FORTRAN control program.

The decoding rules and some decoding techniques are discussed in Section IV. Section V presents a specific simulation problem, recommended user preparation for operating the question-answer system, the terminal session, and the encoded English text description and GPSS program. Conclusions and recommendations for further work are contained in Section VI.

II. OUTLINE OF NLP AND DECODING

NLP may be thought of as a two-part system, the NLP Compiler and the NLP Machine. Both parts are intimately related to a list-structured, record-based memory. This section contains a discussion of the memory, the Compiler, the Machine, and NLP rules for decoding and encoding.

A. NLP MEMORY STRUCTURE AND USE

1. The Cell

The basic unit of memory in NLP is the "cell." Each cell consists of eight bytes (64 bits) of IBM 360/67 computer memory. The cells are sequential elements of a FORTRAN array (named CELL) and are addressed accordingly. A cell may be used as an eight byte word or as a group of four 2-byte words. When used as an eight byte word a cell could represent eight characters in EBCDIC code, a numeric value, or a set of 64 on-off indicators. In the four 2-byte word format the "rightmost" word always contains either a zero or a pointer to (the address of) another cell. The

remaining three words contain data which are normally numerical values, cell addresses, or on-off indicators.

2. Records

A record is a list of cells, and is used to represent an "entity" by holding the values of its "attributes."

For example, a record could represent the punctuation mark ":", or the object "Red; Brick; hardware store; 1125 Market St., Dayton, Ohio; value $50,000; owner J.M. Doe; For sale," or the sentence "The men gave the small boys money."

The number of cells in a record depends on the number of attributes specified. The values of attributes may be actual numerical values or pointers to other records. A special kind of attribute is the indicator (switch value) which, when "turned on," specifies the existence of a particular condition.

For example, a record representing the tires on a vehicle could have a "quantity" attribute with a value of 4, and a "color" attribute with a value of 'black.' The record could also have a "whitewall" indicator to specify that the tires have white sidewalls, and possibly a "tubeless" indicator to specify that they are tubeless. This record could also have a "superset" (SUP) attribute which would indicate that the entities referred to by the record are members of a class of objects called "tire." This record may be input to NLP in the following manner:

```
VEHTIRE(SUP='TIRE',COLOR='BLACK',QUANTITY=4,
        WHITWALL,TUBELESS)
```

where WHITWALL and TUBELESS would have been previously defined as indicators. Other records, representing trucks or automobiles, could point to this record as the value of their "support" attribute. Reference 5 contains a detailed description of the cell and record structure and the manner in which attributes and indicators exist and are referred to within a record.

In general there are three classes of records:

a. Permanent records

Most permanent records are produced by the NLP compiler during initialization of NLP. These records represent items such as decoding rules, encoding rules, objects, attribute names, indicator names, words, etc. Permanent records estabiish a "configuration" of the NLP Machine. Although these records are called permanent, there is a capability for adding or deleting certain of them in the NLP command mode.

b. Semi-permanent records

These records are the end result of the decoding process. Some form of each semi-permanent record exists in the IDS that is ultimately used to encode the output. Semi-permanent records are usually deleted if the user chooses to re-run the NLP Machine using the same or any other configuration.

c. Transient records

Transient records exist only for the purpose of creating other records, or for modifying or adding to the structure of other records. Numerous transient records are created and deleted during the decoding and encoding processes. The term "SEGMENT" record is used in Ref. 5 to refer to transient records.

One unique permanent record, called MEMORY, is the coordination center for the NLP Machine. MEMORY is always directly accessible from both the FORTRAN control program and the decoding and encoding rules. MEMORY was used extensively in this project to communicate between the questionnaire control program and the decoding rules. MEMORY also serves as the starting point for GES and XGES, and the questionnaire. The many uses of MEMORY will become apparent in later sections.

B. NLP COMMAND MODE AND COMPILER

The execution of NLP begins in the command mode. The first user action is to initialize the system from external data sources. External card image files contain rules written in NLP "rule language" format, records that prescribe the initial IDS, and optional lists of attribute and indicator name definitions. These files define a particular NLP Machine starting configuration. When the user specifies a file to be processed, the NLP Compiler translates the input file into an internal record representation of the information in the file. After each file is processed, the user

can cause the current NLP record structure to be dumped onto another external file. This greatly reduces future pre-processing time when the user desires to re-initialize NLP to a particular partial or complete configuration. Other commands available to the user include entering a new file, deleting a rule, defining new attribute names, indicator names, and permanent records, printing out records specified by name or number, and initiating the NLP Machine.

C. NLP MACHINE

Running the NLP Machine is the process of decoding input text into the IDS or encoding output from the IDS. References 5 and 6 describe the GES and XGES encoding systems, and specify an IDS for representing simple queuing problems. For testing GES and XGES, the IDS had to be produced manually and entered from an external file via the NLP Compiler. The end-product of this thesis is a system for producing an encodable IDS from a time-sharing terminal via decoding rules and question answering. While operating the question-answer system, the user "runs" the NLP Machine.

D. NLP RULES

A set of rules (decoding and/or encoding) may be thought of as a "rule language" program for the NLP Machine. Each rule specifies a data structure transformation to be performed when certain conditions are met.

Encoding rules and decoding rules are similar in construction and effect, but are conceptually opposite in application, and are executed (internally) quite differently. In addition, they are processed separately by the NLP Compiler and operate independently under the NLP Machine. The user can not simultaneously decode input and encode output, although the NLP Machine can be easily modified to automatically switch to encoding (from decoding) when a particular IDS condition is sensed.

1. Rule Construction and Meaning

A rule consists of a left part and a right part separated by "-->", the transformation symbol. The data structure changes indicated in the right part of a rule are performed when the conditions of the left part are satisfied. The following is a typical NLP rule:

SEG1(ATTR1.NE.0,IND1,¬IND2) --> SEG2(ATTR1=0,-IND1,IND2)

where SEG1 and SEG2 are names of SEGMENT records. When the rules are initially processed, the NLP Compiler creates a SEGMENT TYPE record for each SEGMENT record name found in the rules. One attribute of a SEGMENT TYPE record is a list of decoding rules which begin with a SEGMENT of that type, e.g.,"SEG1", on the left. "ATTR1" is an attribute name, and "IND1" and "IND2" are indicator names. (If IND1 and IND2 were not initially defined as indicators they would be considered attributes when encountered in the rules, and processed accordingly.) The parenthesized items on the left are condition tests, and on the right are "labeling"

specifications (certain types of condition tests are also permitted on the right). The absence of one or both of the parenthesized expressions means no condition testing (other than the fact that the SEGMENT exists) and/or no labeling operations are required when the rule is executed. The example rule given above can be read:

> "If a SEGMENT record of type SEG1 exists with its ATTR1 attribute not equal (NE) to zero, its IND1 indicator 'on', and its IND2 indicator 'off', then create a new SEGMENT record of type SEG2 and set its ATTR1 attribute to zero, its IND1 indicator to 'off', and its IND2 indicator to 'on'."

Hereafter, the term "SEG2" will be synonomous with the phrase"SEGMENT record of type SEG2."

2. Decoding Rules

Each SEGMENT record created by the NLP Machine via the decoding rules represents the information found in a storing of input text. The relative indices of the first (from the left) and last characters covered by a SEGMENT record are stored as attributes of the record. This feature is employed in the following type of decoding rule to combine adjacent (reading from the left) elements of text:

```
SEG1  SEG2 --> SEG3
```

This rule will create a SEG3, when a SEG1 and a SEG2 exist, that represent consecutive portions of the input character string. If the SEG1 covers character index positions 1-4 and SEG2 covers positions 5-8, then the SEG3 will cover

character positions 1-8. Rule language syntax requires that each element of a rule be separated by one or more blank spaces, and that each rule begin on a new card image line in the file.

During initialization of NLP, a SEGMENT TYPE record is also created for each letter, digit, and punctuation mark. Thus it is possible to write NLP decoding rules which result in the creation of SEGMENT records which represent the meaning of a string of English text. For example, the text:

BLACK CAT

could be processed by the following decoding rules to create a NOUNPH which would cover the text from the B to the T and would represent the information "a four-legged, black animal in the cat family":

```
B L A C K  --> ADJ('BLACK', COLORIND)
C A T      --> NOUN('CAT', ANIMAL, LEGS =4)
ADJ(COLORIND) # NOUN --> NOUNPH(%NOUN,COLOR=SUP(ADJ))
```

In the above rules, on the right, the notation 'BLACK' tells the NLP Machine to set the SUP (superset) attribute of the ADJ record to point to the record named BLACK. Records with names are referred to as "named records." The named record for BLACK could have a SUP attribute pointing to the record named ABSCOLOR, etc. Also, the notation %NOUN means copy all information from the NOUN record into this record; and the # on the left indicates

the presence of a blank space in the input string. Appendix B contains additional information on the syntax and semantics of the decoding rules.

III. THE QUESTION-ANSWER SCHEME

The system developed to meet the objective of this thesis consists of a FORTRAN control program to interact with the user, and a set of NLP decoding rules to interpret user responses and generate an encodable IDS for GES. Each questionnaire session is conducted in three phases, and is initiated by the NLP command "QUESTION:" entered at the terminal. The first part of this section is a description of the general structure of the question-answer scheme. Then each of the three phases of processing are discussed. The emphasis is on the operation of the control program QUEST, a listing of which appears in Appendix D.

A. GENERAL STRUCTURE AND FORMAT

During a question-answer session, many of the computer produced questions and instructions to the user contain data (usually attribute or identification information) provided in previous user responses. Also, the nature of a user response often determines the next computer question or instruction. Certain user responses may be entered in several different forms because the basic decoding scheme scans the input string for the word, phrase, or number values which are appropriate. For example, when the user is prompted with:

'....NAME NEXT STATIONARY ENTITY....OR TYPE "NONE"'

any single word response that was not previously defined, or was defined as a stationary entity name (not previously used), will be decoded as a valid name for a new stationary entity. The subsequent question will then pertain to that stationary entity. The "NONE" part of the above prompt, however, would be satisfied with any of the following responses:

N

NO

NONE

DONE

NO MORE

NO MORE STATIONARY ENTITIES

FINISHED

END

In this case the decoded response (a "done" indication) implies that this phase is complete. Any response that does not constitute a valid stationary entity name or "done" indication would result in an error message sequence being typed at the terminal by the computer.

An error message sequence consists of a reprint of the previous input string with a $ under the last character absorbed by the decoding rules, and (in this case) one of the following messages:

(a) '**INVALID STATIONARY ENTITY NAME'

(b) '**INCOMPLETE REPLY AT $...RE-ENTER'

After an error indication the user is prompted with the same (or a similar) question or instruction. The error detection scheme is discussed in a later section.

In QUEST, each occurrence of the FORTRAN statement "CALL DECODE(ONE,&305)" results in the terminal unlocking for a user response, and the subsequent text decoding (via the rules) being initiated. If decoding is successful (a REPLY is found) and/or the input string is exhausted, a return to QUEST is effected. At this point the logical FORTRAN variables OK (OK=true means "valid response detected") and DONE (DONE=true means a valid "done" response detected) may be examined to determine subsequent action. DONE = true implies OK=true, hence a DONE check (when applicable) always precedes an OK check. OK=false always results in the general error message (b) above, if a specific error message sequence was not initiated from the decoding rules, and a user prompt requesting the same entry. OK= true causes the computer to prompt the user with the next question or instruction. DONE=true (when applicable) is used to indicate the termination of a data entry phase or cyclic sequence and causes the next portion of the questionnaire to be initiated.

One control feature of QUEST is the "checkpoint" (indicated by the parameter: CHKPNT). This signals the decoding system to enable certain additional decoding rules. The FORTRAN statement:

CALL HSTORE(ONE,CHKPNT,MEMORY)

causes the numeric value 1 to be assigned to the CHKPOINT attribute of the IDS record MEMORY. This value can then be accessed from the decoding rules. For example, the rule:

SEG1(CHKPOINT(MEMORY).EQ.1) --> SEG 2

would create a SEG2 only when a SEG1 exists and the CHKPOINT attribute of MEMORY has the value 1.

Additional format and structure features are explained in the following sections.

B. PHASE 1: STATIONARY ENTITIES

A stationary entity in a simulation problem represents any physical element of the system being simulated which does not change in position or usage during the simulation problem. A stationary entity usually represents a point of contact, such as a processing or holding point, between the system and one or more mobile entities which pass through the system. Examples of stationary entities are a gas pump in a service station, a pier in a harbor, a bank, and a teller window in a bank. Phase 1 is the process of identifying and labeling stationary entities.

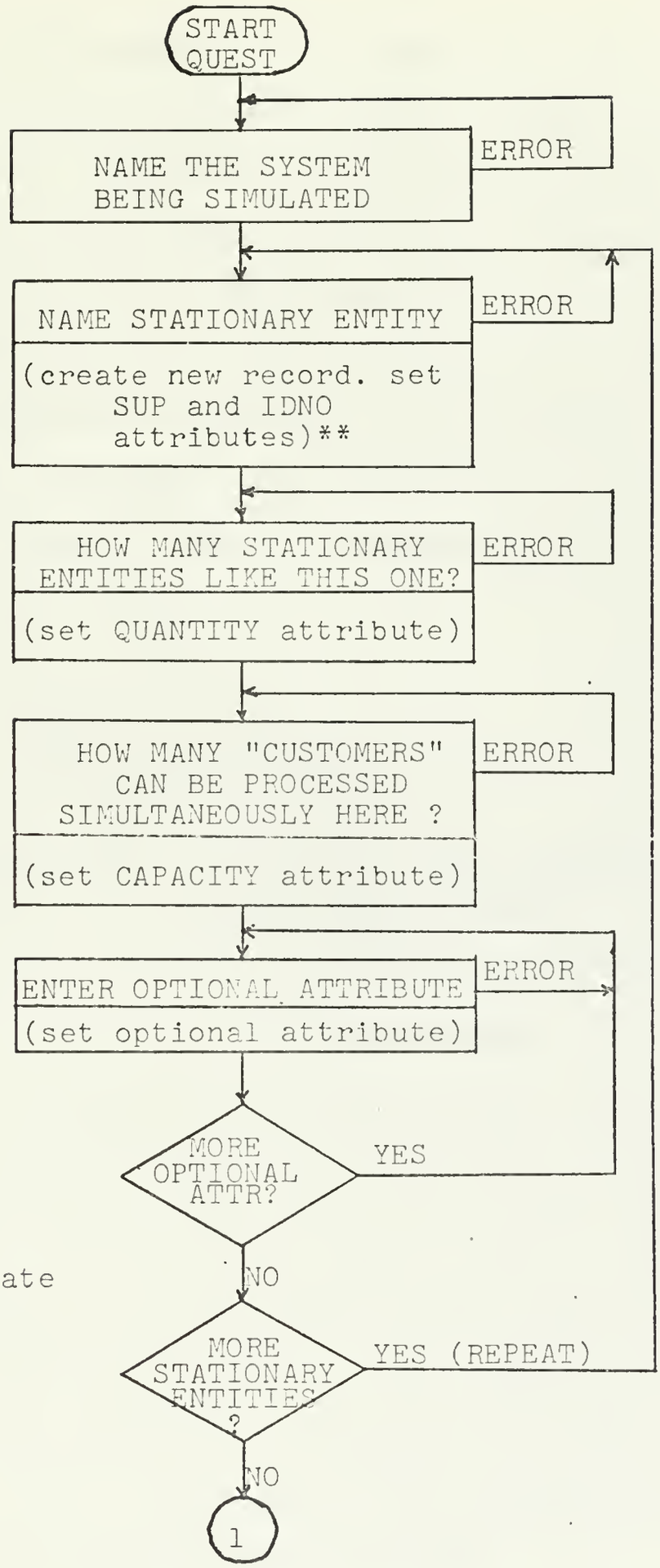
Each stationary entity is represented in the IDS by a record. All stationary entity records except for the one representing the system being simulated have, as a minimum, an identification number (IDNO), a SUP attribute, and a CAPACITY attribute. In addition, they may have an arbitrary

number of "optional" attributes, such as LOCATION, COLOR, etc. Figure 1 shows the functional question-answer sequence from the start of QUEST through complete definition of all stationary entities. Each action (defined in Section III-D) is considered to occur at or within a stationary entity. Therefore, each action has a LOCATION attribute that points to a stationary entity record. Stationary entity records contain the information needed by GES to create the corresponding GPSS Storage or Facility.

C. PHASE 2: MOBILE ENTITIES

A mobile entity in a simulation problem represents the dynamic element that is processed or serviced by the system being simulated. Mobile entities can be thought of as entering, being processed in one or more stages, and leaving the system. In a GPSS program, mobile entities are represented by "transactions." Examples of mobile entities are a customer in a bank or store, an automobile in a gas station, and a ship in a harbor. Phase 2 is the process of identifying and labeling mobile entities.

The question-answer decoding system creates IDS records for mobile entities. These records contain the data needed by GES to produce a GPSS program that generates representative transactions. Every mobile entity record contains, as a minimum, SUP and IDNO attributes. The question-answer system allows the user to define a maximum of one level of sub-structure for each different (by SUP attribute)



** Parentheses indicate functions of the decoding rules.

Figure 1. Stationary Entity Entry Sequence

mobile entity. If a mobile entity record has a sub-structure, it also has a classification (CLASATR) attribute which points to the attribute that delineates the sub-structure. Each sub-structure record has a different value for that attribute, plus a STRUC attribute that points to its super-structure record. Each sub-structure record also has a QUANTITY attribute specifying the percentage of mobile entities of that particular type that belong to this sub-classification. Figure 2 gives an example of this relationship.

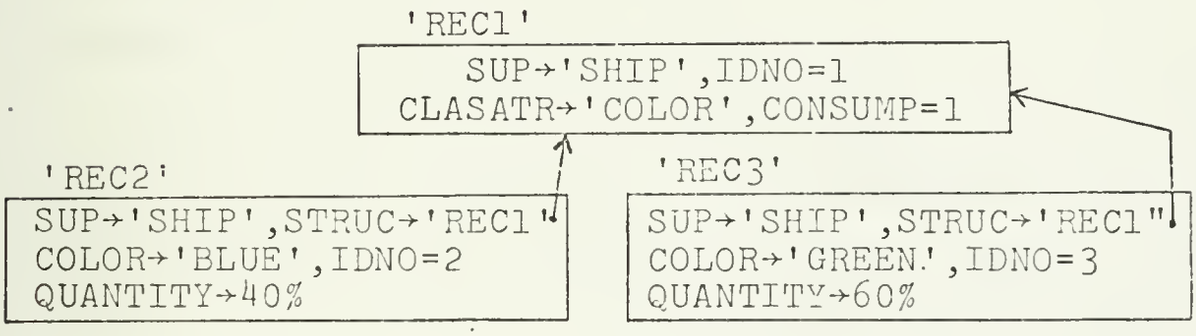


Figure 2. Sub-Structure Relationship

The "→" symbol in Figure 2 means the attribute on the left points to (contains the address of) an IDS record which is used to represent the data on the right. The "=" symbol means the attribute on the left contains the actual value on the right. The CONSUMP attribute of REC1 indicates the number of service units each 'SHIP' requires at a processing point. Figure 3 shows, functionally, the question-answer sequence for creating mobile entity records.

D. PHASE 3: ACTION SEQUENCE

There are two types of actions that can occur in a simulation problem, an event or an activity. An event is an action that does not consume time within the simulated system. A car arrives at a gas station, or a customer leaves the store, are typical examples of events. Activities, on the other hand, are actions which consume simulated time, such as: a car is serviced at the gas pump, or a blue ship unloads cargo at the dock. Records representing actions and their interrelationships form the heart of the structure from which GES encodes a GPSS program. Phase 3 is the process of developing this structure.

Every action record for an activity contains the following essential attributes: SUP (the name of the activity), SUCC (pointer to the "successor" action record), MBNTY (pointer to the record for the mobile entity involved in this activity), LOCATION (pointer to the stationary entity where the activity takes place), and DURATION (describes the activity time duration, usually via a probability distribution). Events may or may not appear as action records. The arrival and departure of mobile entities are events which are represented by action records. In this case, all attributes listed above for activities, except DURATION, are required. The action record for arrivals must, however, contain an inter-arrival time (IETM) attribute. An event not explicitly represented by an action record is typically

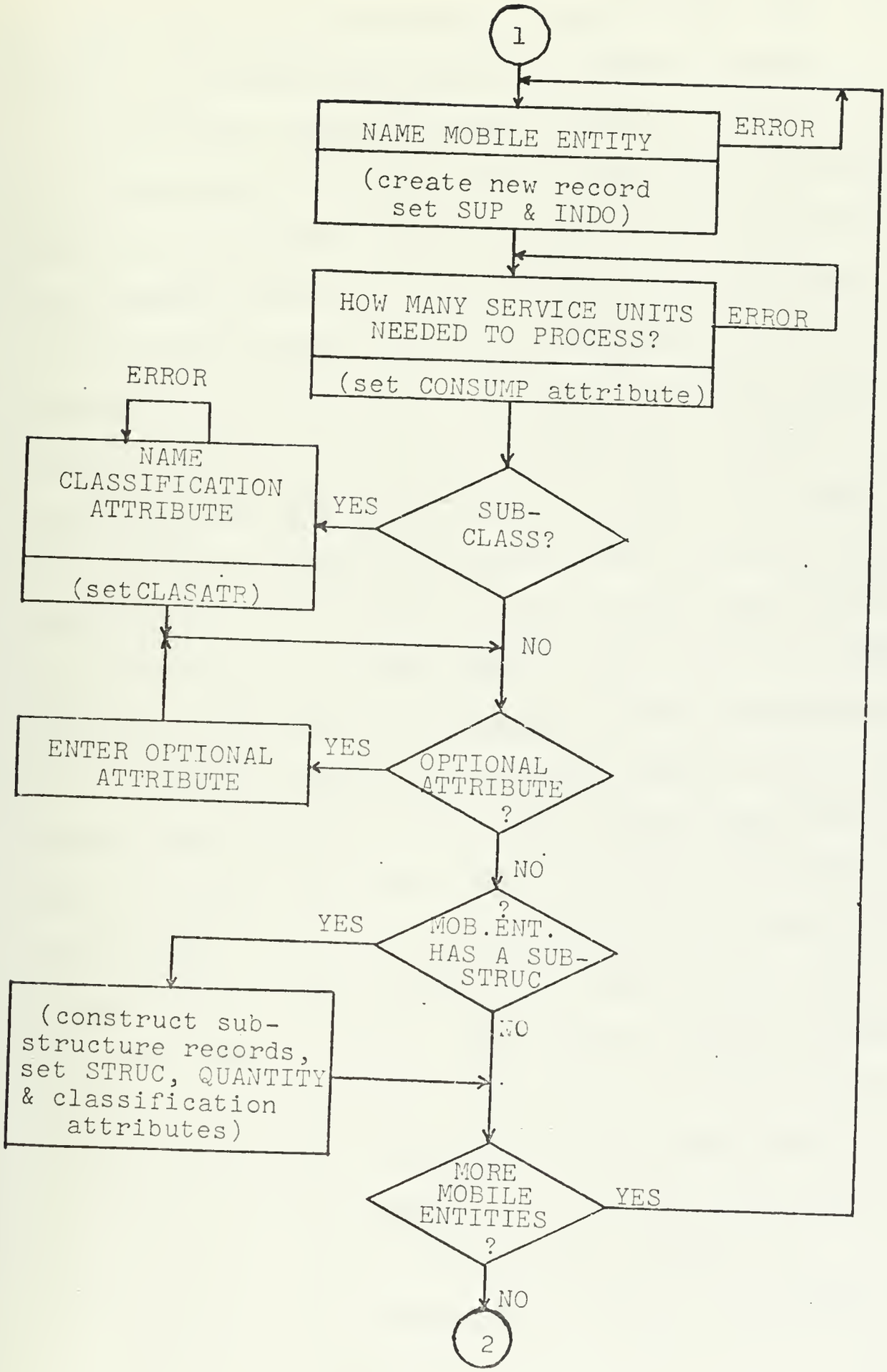


Figure 3. Mobile Entity Sequence

one which yields a GPSS Function and TRANSFER block, and appears as the SUCC attribute of an action record.

Developing a question-answer decoding procedure that would produce an encodable IDS for the action sequence proved to be the most difficult part of this research. The procedure chosen is closely related to the action list (an IDS record which is a list of pointers to the action records). To initiate the sequence, an action record (ARRIV) for the arrival of the first-entered mobile entity is automatically generated by the decoding rules. The user is instructed to specify an inter-arrival time probability distribution for all mobile entities and a basic time unit for the simulation. The decoding rules enter the ARRIV record on the action list without a SUCC (successor) attribute. The basic action-builder loop starts with the decoding rules picking up the first action (from the action list) that does not have a SUCC attribute and is not a LEAVE event. The user is then prompted to enter information concerning the next action. The result will be one or more new action records, each without a SUCC attribute, and each entered on the action list. In the process, the action record which started the cycle will receive a SUCC attribute, and the stage is set for the next cycle. In essence, the question the user must answer is:

> "What does the mobile entity <name 1> do after the action <act 1> is complete?"

If the response sequence results in a single new action record, the cycle simply repeats. However, the user might wish to say:

"After <act 1> , 30% of the <name 1>'s go to <act 2> , 40% go to <act 3> , and 30% go to <act 4>."

This response would be entered as a "conditional" (discussed later). The problem here is that <act 2>, <act 3>, and <act 4> would not have been defined or identified yet, and the action just specified is an event that does not yield an action record. To solve this problem, the concept of a "temporary action" (TEMPACTN) was developed. A TEMPACTN record is created by the decoding rules in every case where an action must be defined or identified later. Each TEMPACTN record is added to the action list without a SUCC attribute, and is filled with reference information. After a TEMPACTN is processed by the action-builder, it is always deleted from the IDS, and its position in the action list may be filled by an action record. The action-builder sequence terminates when every action on the action list has a SUCC attribute or is a LEAVE event, and all mobile entity types have been processed.

Figure 4 is an overall flow diagram of the action-builder sequence. Another way to follow this procedure in more detail is to examine the ACTION LOOP portion of subroutine QUEST in Appendix D.

Special GPSS actions such as:

```
TRANSFER     ,FN7 (branch according to function 7)
GATE NU      1    (hold here until facility 1 is not
                   in use)
```

can not be conveniently entered in the same manner as the action:

"service customers from a queue, service time is exponentially distributed, mean=12 minutes."

Hence the concept of a "conditional" was used. Conditionals require key-word entries from the terminal which activate specific decoding rules and cause unique questions or instructions to be issued to the user. The implementation of conditionals is general and can be conveniently expanded both in QUEST (via the computed GO TO statement) and in the decoding rules under a unique checkpoint value (using the rule test: CHKPOINT(MEM).EQ.5). Initially, only the two conditionals mentioned above were implemented.

The mobile entity sub-structure scheme described in Section III-C provides an additional capability while constructing the action sequence. For example, in the case where the first mobile entity record represents 'SHIP' and has a CLASATR→'COLOR', the user would be prompted in the following manner (after the ARRIVE action record was completely built by the decoding rules):

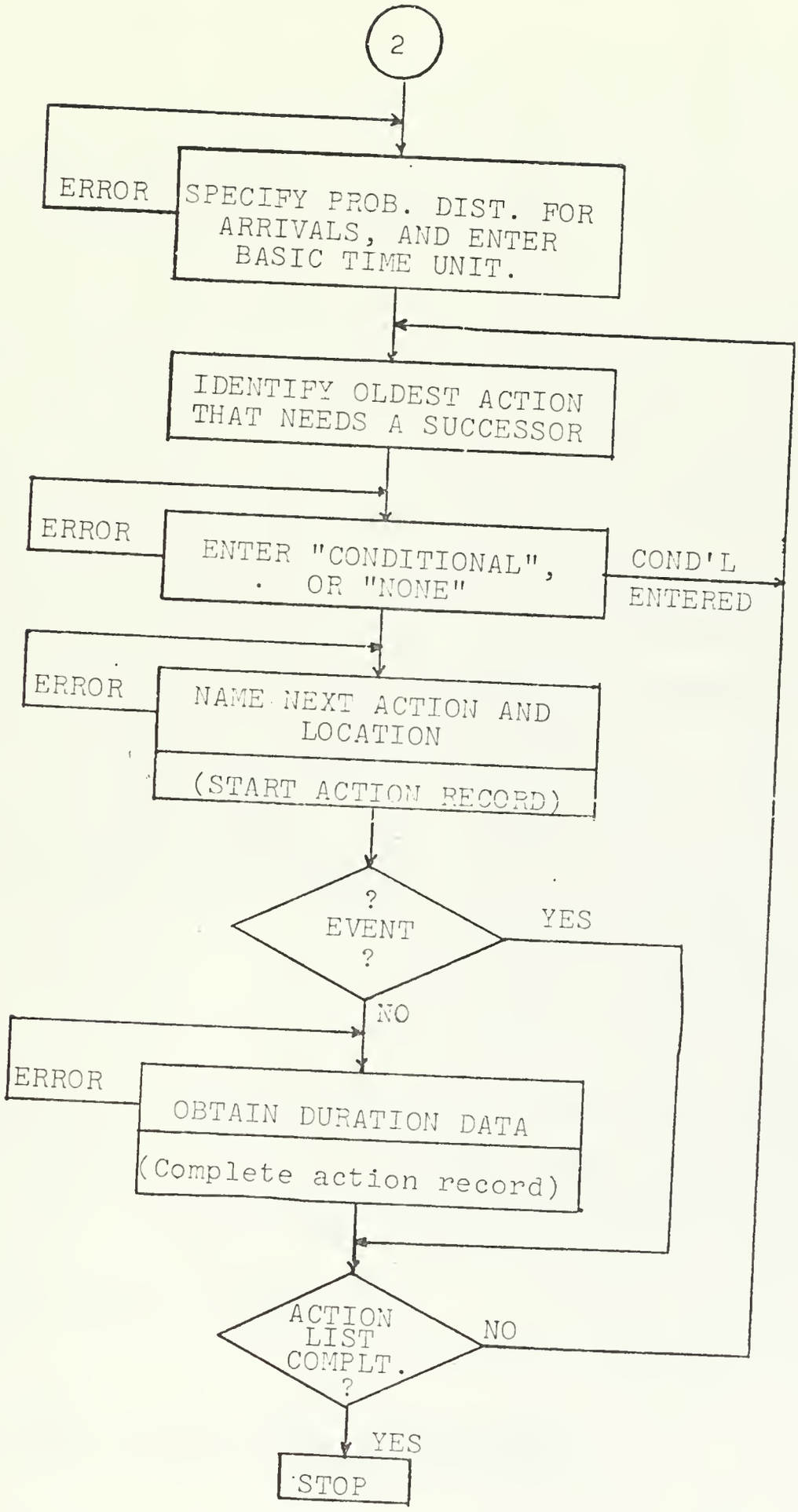


Figure 4. Action Builder Sequence

```
.
.
.
.
FOLLOWING THE ACTION:
SHIP    ARRIV    AT      PORT
DOES ROUTING DEPEND ON COLOR OF SHIP   ?

<response>
```

If the user answers 'YES' then the decoding rules automatically create "function definition" records and a TEMPACTN record for each 'SHIP' mobile entity record in the sub-structure (each record with SUP→'SHIP' and a STRUC attribute). Function definition records are ultimately encoded by GES into GPSS functions that are used to assign a transaction parameter, or determine transaction routing, etc. A 'YES' response would then be followed by:

```
FOLLOWING THE ACTION:
BLUE    SHIP    ARRIV    AT    PORT
<CONDITIONAL option prompt>

<response>
.
.
.
.
```

as a result of a TEMPACTN record. In this case, the MBNTY of the TEMPACTN is the sub-structure 'SHIP' record with COLOR→'BLUE'. In the course of defining the next action the user would also be asked:

```
.
.
.
.
WILL NEXT ACTION APPLY TO ALL SHIP
                  VICE BLUE SHIP ONLY?

<response>
```

This determines the MBNTY attribute for the resulting action record. If the answer is 'YES' the MBNTY attribute points to the original (top level) 'SHIP' record. Also, if the MBNTY of the action 'UNLOAD' has a sub-structure, and the user is at the point of defining a duration for the action, the following question is asked:

.
.
.
.

DOES DURATION OF UNLOAD DEPEND ON COLOR OF SHIP ?

<response>

A 'YES' answer would cause the decoding rules to create function definition records for the selection of 'UNLOAD' time durations based on 'COLOR'.

When the action-builder sequence is finished, the user is prompted to enter a value for total time to be simulated. The IDS is then complete and the computer prints:

DATA STRUCTURE FOR PORT SIMULATION IS COMPLETE

QUEST then terminates and user control is returned to the NLP command level.

IV. THE QUESTION-ANSWER DECODING RULES

NLP decoding rules are used to specify the processing of user responses needed to produce IDS records. General construction and application features of NLP decoding rules are discussed in Section II-D and in Appendix A. The purpose of this section is to describe the decoding rule

implementation for the question-answer decoding system. Appendix B contains a complete listing of these decoding rules.

A. GENERAL FEATURES

As mentioned before, a decoding rule can communicate with the FORTRAN program via the special record MEM (an acceptable rule-language abbreviation of MEMORY). This capability is used extensively in the question-answer decoding system. The following are examples of its usage:

(1)"...,CHKPOINT(MEM).LT.5,..." This is a condition test that allows further processing of this rule only if the value of the CHKPOINT attribute of MEM is less than 5. This rule execution control attribute is used frequently by QUEST.

(2)"...,ERR(MEM)=17,..." This causes MEM to have an ERR attribute, if not already present, and assigns it a value of 17. Attribute values can be extracted and examined in the FORTRAN program by use of the HVAL function subroutine of NLP, if the SEGMENT record address is known (MEMORY is always known) and the attribute identification number is known (normally pre-defined, for the rules and FORTRAN, in the initial NLP machine configuration).

Another method of rule-FORTRAN communication is by the use of "ROUTINES." When used, ROUTINE names and identifying numbers must be defined before the NLP Compiler processes the rules. While running the NLP Machine, if a ROUTINE

name is encountered in parentheses during execution of a rule, a set of FORTRAN statements is executed according to the identifying number associated with the ROUTINE. In the question-answer decoding system, for example, the name ROK is defined as ROUTINE 21. The corresponding FORTRAN statements cause the logical variable OK to be set to "true." The state of OK is frequently tested by the control program QUEST. Appendix E is a listing of the ROUTINES that were written for the question-answer system.

B. THE DECODING SCHEME

In general, the question-answer decoding rules first parse the input string into SEGMENT records representing words and numbers, then into more comprehensive and specific SEGMENT records based on the type and relationship of the previous records. Many other conditions, such as CHKPOINT value, questionnaire phase, attributes and indicators in IDS records, etc., are used to determine the correct parse sequence for an input string. The goal is to appropriately add the information in an input string to the IDS. The question-answer decoding rules were designed to allow a great amount of flexibility in the input string by detecting key words or phrases.

1. Record Creation and Labeling

Each time the decoding process is begun by calling DECODE from QUEST, the result is either the creation of a new semi-permanent IDS record or the modification of an existing IDS record. To complete an IDS record for, say,

a mobile entity, the user must respond at least four times. Each response completely re-initiates the decoding process, which presents the problem of remembering the address and status of the current mobile entity record and determining which rules to apply. The problem was solved in part by the use of the attribute CURSEG (current segment record) of MEMORY. When a mobile entity name is entered at the correct response point in QUEST, the basic IDS record is created and CURSEG(MEM) is set to point to this record. The rules applied for subsequent user responses have access to this record via CURSEG(MEM) and can test conditions, label, etc., accordingly. There are numerous examples of the use of CURSEG(MEM) in the rules of Appendix B.

2. Naming Entities, Actions, and Attributes

The decoding rules allow for arbitrary naming of stationary entities, mobile entities, actions, and optional attributes. This is accomplished by a series of rules that pick up arbitrary character strings. For each segment of any input string that begins with an alphabetic character and is properly delimited by blank spaces or other allowable punctuation marks, a segment of type WORD is eventually created with the SUP attribute set to the arbitrary name. Although the name may have up to 80 characters, the record for the SUP attribute will contain only the first eight characters. This general word accumulation process is accomplished with two ROUTINE's. The first, RGETJ, stores the index of the first character of the WORD as the JNUM

attribute of the WORDP (word part) segment record that is initially created. When the tail end delimiting character (PUNCB) occurs, a WORD segment is created and the routine RSETSUP, using the value of JNUM, sets the SUP attribute as described above. In a sense, the system "learns" new words or names, and classifies them as "unknown" until their context is determined. Future reference to any name must comply with its original usage or (for pre-entered names) pre-defined subset classification. Numerical values are assembled in a similar manner although no ROUTINE's are needed.

3. REPLY Segments

The final SEGMENT TYPE that is formed when a satisfactory user response is decoded is the REPLY. When a REPLY is detected, or the end of the input string is sensed and no more decoding rules apply, a return to QUEST is effected and the user is prompted with a new question or instruction, or receives an appropriate error notification.

4. Error Detection Capability of the Rules

In addition to the error detection capability mentioned in Section III, some error message sequences are initiated by the decoding rules. This is accomplished by RERR (a ROUTINE) and the following rule language notation:

xxxxx(.....,ERR(MEM)=17,RERR,....)

The ROUTINE RERR extracts the value of the ERR attribute of MEMORY and calls the FORTRAN sub-program ERROR with that value (the error number). RERR is normally used to notify

the user of illegal use of a previously defined name. 'BLUE' could not, for instance, be used as anything other than the value of a COLOR attribute. This type of error also causes the terminal to print the super-set (SUP) "chain" for the illegal name. (To examine or traverse the SUP "chain" means to look at the value of the SUP, then the value of the SUP of the SUP, then the value of the SUP of the SUP of the SUP, etc., until the desired value is located or the chain is exhausted.) In this example, if 'BLUE' were entered as a mobile entity name, the user would receive the following error message reply:

```
    .
    .
    .
    .
BLUE                                  }
    $                                 }  error message
**INVALID MOBILE ENTITY NAME          }
BLUE    → ABSCOLR                     }
ABSCOLR → QUALVAL                     } SUP "chain"
QUALVAL → VAL                         }
RE-ENTER NAME OF MOBILE ENTITY

<response>
```

The FORTRAN subroutine PRTCLS (included in Appendix D) is used to print the SUP "Chain."

5. Optional Attributes

During the entry of each mobile and stationary entity, the user is given an opportunity to enter an unlimited number of optional attributes. At this point, any attribute not previously defined for that record may be entered. Each entry requires an attribute identification name and value (name or number). Errors result when

pre-defined names (identification or value) are not correctly used. The following is an example of variations in user response that are acceptable:

```
COLOR BLUE

COLOR=BLUE

COLOR is BLUE

THE COLOR OF THE SHIP IS BLUE
```

An example of an invalid optional attribute entry and error sequence is:

```
.
.
.
ENTER OPTIONAL ATTRIBUTE FOR SHIP

COLOR IS LARGE
                $
**ATTRIBUTE IS NOT IN CLASS INDICATED
ENTER OPTIONAL ATTRIBUTE FOR SHIP

<response>
```

The applicable rules recognize that the pre-defined word LARGE is not a legal value of the attribute COLOR. However, the rules will accept any "unknown" word as a value of the attribute COLOR and would set the SUP of the unknown word to 'ABSCOLR'. In this manner the system is capable of "learning" and classifying new names at virtually any point where name entries are required.

6. Action Generation

As pointed out in Section III-D, the decoding rules search the action list for the oldest action description record which does not have a SUCC (successor) attribute. The attribute LASTACT(MEM) becomes a pointer to this action

record, and the MBNTY (mobile entity) attribute of the action record becomes the value of MBNTY(MEM). Using the action record LASTACT(MEM), the rules then have access to the actual characters used by QUEST to print the line:

BLUE SHIP ARRIV AT PORT

These words are set as the values of the pre-specified attributes A1 through A5 of MEMORY, and are therefore available at the FORTRAN level. This communication technique is the means by which the user is guided systematically through the question-answer session. A much more elegant communication technique, however, would be to use the existing NLP Machine encoding capability and encode grammatically correct English sentences. This would require an additional set of encoding rules, and remains an area of possible expansion of the basic question-answer system.

The action generating rules create and assemble the action records, distribution records, "conditional" description records, etc., needed to describe a simulation problem in accordance with the IDS required by GES. All actions are prescribed by the user, but the strict question-answer format ensures that an encodable IDS will be generated. In general the user can cause mobile entities to be processed by the system in any desired sequence, including routing to formerly defined actions. However, the system does not provide a check against user specification of an endless loop for a mobile entity. Additionally, the system does not allow the user to specify an arbitrary probability

distribution, but instead provides a choice (when appropriate) among normal, exponential, and uniform, and asks him to provide the necessary parameters (mean, standard deviation, range). In lieu of a probability distribution, the user may specify a constant value of, for example, inter-arrival times of mobile entities.

When the action list search determines that all action records are completed, the user is asked to supply the total time for which the simulation is to be run, then notified that the IDS representing the simulation problem is complete. At this point the user may elect to encode an English language description of his simulation problem (how the computer "sees" it) using the NLP encoding scheme developed by McGee [Ref.6]. If this description is satisfactory, he may then encode the GPSS simulation program using GES or XGES.

V. SAMPLE PROBLEM

Figure 5, extracted from Ref.5, is an example of a queuing system simulation problem. Figure 6 shows a suggested form, to be prepared prior to using the question-answer system, that includes the minimum data needed to describe stationary and mobile entities. The form in Figure 6 is filled in according to the problem description in Figure 5. Figure 7 is a flow diagram of the same problem. This diagram includes the responses (to questions or

There is a port containing a harbor, 3 docks, 2 piers, a depot, and a barge. Ships arrive at the port with an interarrival time of 5 hours, uniformly distributed, with a range of ± 1 hour. 50% of the ships are blue ships, 30% are red, and 20% are green. After a ship arrives at the port, it unloads cargo at any available dock. Each dock has a capacity of 1 unit. Each ship takes up 1 unit of capacity. Unloading time at the dock is normally distributed as follows:

blue ship - mean of 5 hours, std dev of 1.5 hours
red ship - mean of 4 hours, std dev of 1.0 hours
green ship - mean of 3 hours, std dev of .5 hours

After unloading at a dock, a blue ship unloads cargo at the barge, a red ship unloads cargo at the depot, and a green ship unloads cargo at a pier. The barge has a capacity of 1 unit, a pier has a capacity of 1 unit, the depot has a capacity of 4 units. Unloading times are as follows:

barge - 1.5 hours, exponentially distributed
depot - 1 hour, exponentially distributed
pier - 1 hour, normally distributed, std dev of 15 minutes

Next, after these latest unloadings, 40% of the ships load cargo at a dock, and the remainder wait in the harbor. Dock loading time is 2 hours for any ship. After loading cargo at a dock, a ship waits in the harbor. A ship waits in the harbor until the barge is unoccupied. After waiting in the harbor, a ship leaves the port. The basic time unit is the minute. Problem duration is 4 days.

Figure 5. Port Facility Simulation Problem

instructions) that the user will be expected to supply. A diagram of this sort is extremely helpful for effective use of the question-answer system.

Figure 8 is an actual terminal session showing the computer questions and instructions, and the user responses for this sample problem. Figures 9 and 10 (extracted from Ref.6) show the computer generated English description of this problem and the resulting GPSS program. The output produced by this GPSS program may be found in Ref.6.

SYSTEM NAME: Port

BASIC TIME UNIT: Minute

TOTAL PROBLEM TIME: 5760min.

STATIONARY ENTITIES:

NAME	QUANTITY	CAPACITY	OPTIONAL ATTRIBUTES	
			IDNAME	NAME/VALUE
DOCK	3	1	LOCATION	PORT
PIER	2	1	"	"
DEPOT	1	4	"	"
BARGE	1	1	"	"
HARBOR	1	UNLIMITED	"	"

MOBILE ENTITIES:

NAME	STRUCTURE LEVEL	SERVERS NEEDED	SUB-CLASS ATTRIBUTE	LEVEL 2 VALUE	%	OPT. ATTRIBTS
Ship	1	1	Color	—	100	—
Ship	2	1	—	Blue	50	—
Ship	2	1	—	Red	30	—
Ship	2	1	—	Green	20	—

Figure 6. Stationary and Mobile Entity Data Form

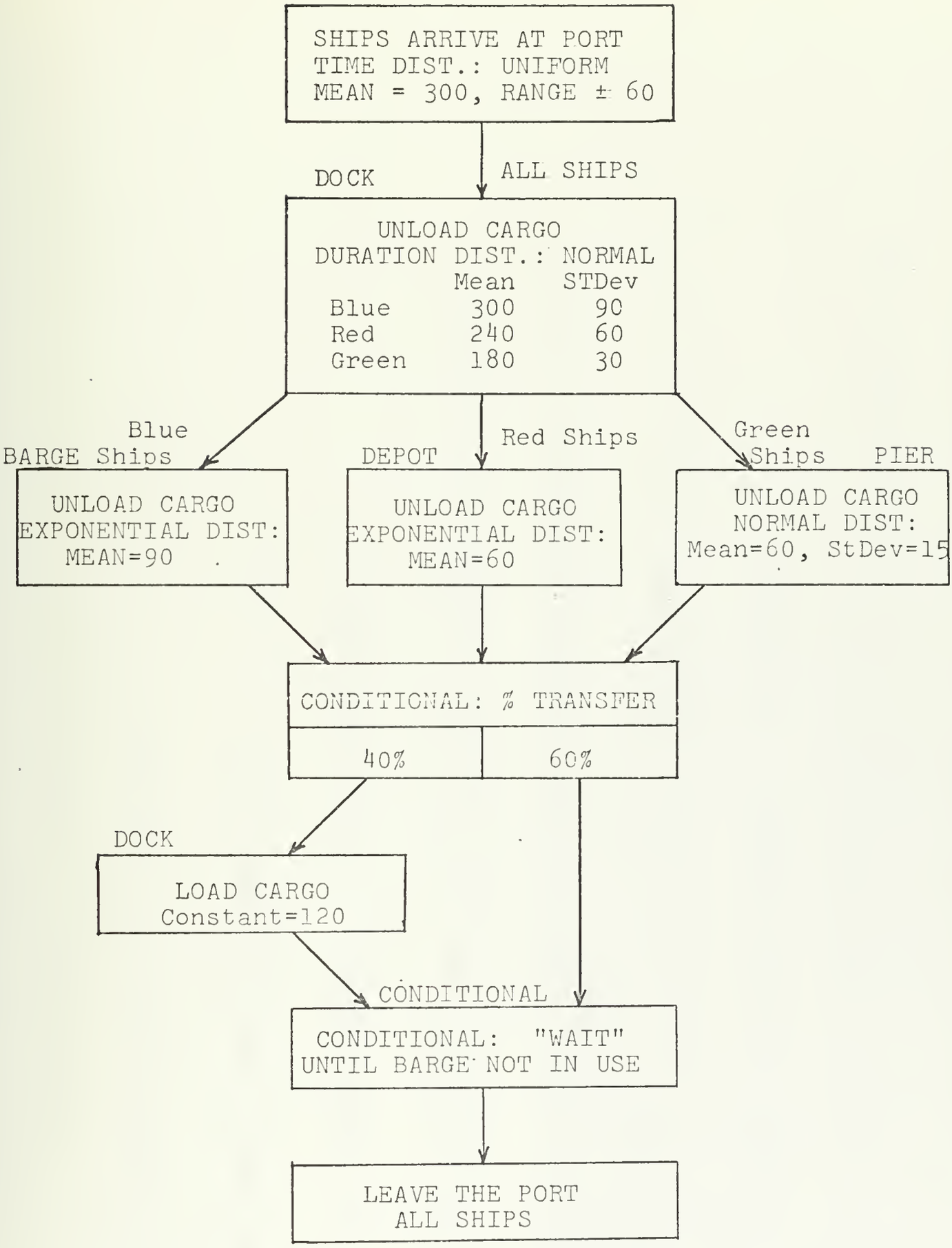


Figure 7. Problem Flow Diagram (Action Sequence)

```
question:

ENTER NAME FOR SYSTEM TO BE SIMULATED

port

ENTER NAME OF FIRST STATIONARY ENTITY

dock

HOW MANY IDENTICAL DOCK      ENTITIES ARE THERE ?

3

HOW MANY "CUSTOMERS" (MAX) CAN EACH DOCK      PROCESS SIMULTANEOUSLY ?

1

IDENTIFY AND ASSIGN VALUES OR NAMES TO OPTIONAL ATTRIBUTES; EXAMPLES:
  LOCATION IS THE HARBOR
  WEIGHT = 1000 POUNDS
  SPEED IS FAST
ENTER OPTIONAL ATTRIBUTE FOR DOCK      (OR "NONE")

location = port

ENTER OPTIONAL ATTRIBUTE FOR DOCK      (OR "NONE")

none

DOCK      COMPLETE...NAME NEXT STATIONARY ENTITY OR TYPE "NONE" IF DONE

pier
```

Figure 8. Terminal Session for Port Facility Problem

```
HOW MANY IDENTICAL PIER     ENTITIES ARE THERE ?
2
HOW MANY "CUSTOMERS" (MAX) CAN EACH PIER     PROCESS SIMULTANEOUSLY ?
1
ENTER OPTIONAL ATTRIBUTE FOR PIER     (OR "NONE")
location, port
ENTER OPTIONAL ATTRIBUTE FOR PIER     (OR "NONE")
none
PIER     COMPLETE...NAME NEXT STATIONARY ENTITY...OR TYPE "NONE"
depot
HOW MANY IDENTICAL DEPOT     ENTITIES ARE THERE ?
1
HOW MANY "CUSTOMERS" (MAX) CAN EACH DEPOT     PROCESS SIMULTANEOUSTL ?
4
ENTER OPTIONAL ATTRIBUTE FOR DEPOT     (OR "NONE")
the depot is located in the port
```

Figure 8. (CONTINUED)

```
ENTER OPTIONAL ATTRIBUTE FOR DEPOT    (OR "NONE")
none
DEPOT    COMPLETE...NAME NEXT STATIONARY ENTITY...OR TYPE "NONE"
barge
HOW MANY IDENTICAL BARGE    ENTITIES ARE THERE ?
1
HOW MANY "CUSTOMERS" (MAX) CAN EACH BARGE    PROCESS SIMULTANEOUSLY ?
1
ENTER OPTIONAL ATTRIBUTE FOR BARGE    (OR "NONE")
location is in the port
ENTER OPTIONAL ATTRIBUTE FOR BARGE    (OR "NONE")
none
BARGE    COMPLETE...NAME NEXT STATIONARY ENTITY...OR TYPE "NONE"
harbor
HOW MANY IDENTICAL HARBOR    ENTITIES ARE THERE ?
1
HOW MANY "CUSTOMERS" (MAX) CAN EACH HARBOR    PROCESS SIMULTANEOUSLY ?
99999
```

Figure 8. (CONTINUED)

```
ENTER OPTIONAL ATTRIBUTE FOR HARBOR   (OR "NONE")

location is the port

ENTER OPTIONAL ATTRIBUTE FOR HARBOR   (OR "NONE")

none

HARBOR   COMPLETE...NAME NEXT STATIONARY ENTITY...OR TYPE "NONE"

none

ENTER NAME OF FIRST MOBILE ENTITY

ship

HOW MANY SERVICE UNITS DOES EACH SHIP     REQUIRE ?

1

IS THERE A SUB-CLASSIFICATION OF SHIP     (MAX OF ONE) ?

yes

NAME THE ATTRIBUTE THAT DISTINGUISHES EACH SHIP

color

OPTIONAL ATTRIBUTES FOR ALL SHIP     REGARDLESS OF COLOR
ENTER OPTIONAL ATTRIBUTE FOR SHIP     (OR "NONE")

none
```

Figure 8. (CONTINUED)

```
NOW ENTER COLOR    DATA UNIQUE TO EACH SHIP
ENTER NAME OR VALUE OF COLOR    FOR SHIP      NUMBER    1

blue

WHAT PERCENTAGE OF SHIP      HAVE COLOR      = BLUE      ?

50

ENTER NAME OR VALUE OF COLOR    FOR SHIP      NUMBER    2

red

WHAT PERCENTAGE OF SHIP      HAVE COLOR      = RED       ?

30

ENTER NAME OR VALUE OF COLOR    FOR SHIP      NUMBER    3

green

WHAT PERCENTAGE OF SHIP      HAVE COLOR      = GREEN     ?

20

SHIP      COMPLETED..NEXT MOBILE ENTITY NAME..OR "NONE"

none
```

Figure 8. (CONTINUED)

```
CONSTRUCT ACTION SEQUENCE FOR PORT
PROBABILITY DISTRIBUTIONS AVAILABLE: EXPONENTIAL, UNIFORM, NORMAL, CONSTANT
 ENTER TYPE OF DISTRIBUTION FOR TIME BETWEEN ARRIVALS OF SHIP     AT THE PORT

uniform

ENTER AVERAGE VALUE(MEAN) OF UNIFORM DISTRIBUTION

300

GIVE STANDARD DEVIATION (NORMAL), OR PLUS/MINUS RANGE VALUE (UNIFORM)

60

GIVE THE TIME UNIT TO BE USED (HOUR,SECOND,ETC.)

minute

FOLLOWING THE ACTION:
SHIP    ARRIV   AT      PORT
DOES ROUTING DEPEND ON COLOR    OF SHIP    ?

no

FORMAT FOR CONDITIONAL SPECIFICATION:
  1. <TYPE OF CONDITIONAL>,<IDENT NAME> FOR NEW CONDITIONAL
  2. <IDENT NAME> ONLY FOR ROUTE TO KNOWN CONDITIONAL
  3. "NONE" IF NONE
SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

none
```

Figure 8. (CONTINUED)

```
NAME NEXT ACTION AND LOCATION FOR SHIP

unload at the dock

ENTER DISTRIBUTION TYPE FOR DURATION OF UNLOAD

normal

DOES DURATION OF UNLOAD   DEPEND ON COLOR    OF SHIP     ?

yes

FOR NORMAL DIST. OF UNLOAD  BLUE    SHIP     GIVE: MEAN, STANDARD DEVIATION

300, 90

FOR NORMAL DIST. OF UNLOAD  RED     SHIP     GIVE: MEAN, STANDARD DEVIATION

240, 60

FOR NORMAL DIST. OF UNLOAD  GREEN   SHIP     GIVE: MEAN, STANDARD DEVIATION

180, 30

FOLLOWING THE ACTION:
SHIP    UNLOAD  AT      DOCK
DOES ROUTING DEPEND ON COLOR    OF SHIP     ?

yes

FOLLOWING THE ACTION:
BLUE    SHIP    UNLOAD  AT      DOCK
SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

none
```

Figure 8. (CONTINUED)

```
WILL NEXT ACTION BE FOR ALL SHIP     VICE BLUE     SHIP     ONLY ?

no

NAME NEXT ACTION AND LOCATION FOR BLUE     SHIP

unload at the barge

ENTER DISTRIBUTION TYPE FOR DURATION OF UNLOAD

exponential

GIVE MEAN OF EXPONENTIAL DIST. FOR DURATION OF UNLOAD  BLUE     SHIP

90

FOLLOWING THE ACTION:
RED      SHIP     UNLOAD  AT       DOCK
SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

none

WILL NEXT ACTION BE FOR ALL SHIP     VICE RED      SHIP     ONLY ?

no

NAME NEXT ACTION AND LOCATION FOR RED      SHIP

unload at the depot

ENTER DISTRIBUTION TYPE FOR DURATION OF UNLOAD

expon
```

Figure 8. (CONTINUED)

```
GIVE MEAN OF EXPONENTIAL DIST. FOR DURATION OF UNLOAD  RED     SHIP

60

FOLLOWING THE ACTION:
GREEN   SHIP    UNLOAD  AT      DOCK
SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

none

WILL NEXT ACTION BE FOR ALL SHIP     VICE GREEN   SHIP     ONLY ?

no

NAME NEXT ACTION AND LOCATION FOR GREEN   SHIP

unload at pier

ENTER DISTRIBUTION TYPE FOR DURATION OF UNLOAD

normal

FOR NORMAL DIST. OF UNLOAD  GREEN   SHIP     GIVE: MEAN, STANDARD DEVIATION

60, 15

FOLLOWING THE ACTION:
BLUE    SHIP    UNLOAD  AT      BARGE
SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

transfer, split1

MIGHT SHIP     OTHER THAN BLUE    SHIP     PASS THROUGH SPLIT1  ?

yes
```

```
ENTER PERCENT OF EACH BRANCH FROM SPLIT1
   EXAMPLE: 30,30,40

40,60

FOLLOWING THE ACTION:
RED     SHIP    UNLOAD  AT      DEPOT
SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

split1

FOLLOWING THE ACTION:
GREEN   SHIP    UNLOAD  AT      PIER
SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

split1

FOLLOWING THE ACTION:
SHIP    DEPARTS THE EVENT NAMED SPLIT1
VIA THE 40% LEG, (ROUTE 1)
DOES ROUTING DEPEND ON COLOR    OF SHIP    ?

no

SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

none

NAME NEXT ACTION AND LOCATION FOR SHIP

load at the dock
```

Figure 8. (CONTINUED)

```
DOES DURATION OF LOAD     DEPEND ON COLOR    OF SHIP    ?

no

ENTER DISTRIBUTION TYPE FOR DURATION OF LOAD

constant

ENTER CONSTANT LOAD     TIME FOR SHIP

120

FOLLOWING THE ACTION:
SHIP    DEPARTS THE EVENT NAMED SPLIT1
VIA THE 60% LEG, (ROUTE 2)
DOES ROUTING DEPEND ON COLOR   OF SHIP    ?

no

SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

wait, wait1

ENTER: LOCATION OF WAIT, ENTITY CAUSING DELAY, AND CONDITION NEEDED TO CONTINUE

harbor, barge, not in use

FOLLOWING THE ACTION:
SHIP    LOAD    AT      DOCK
DOES ROUTING DEPEND ON COLOR    OF SHIP    ?

no
```

Figure 8. (CONTINUED)

```
SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

wait1

FOLLOWING THE ACTION:
SHIP    WAIT    AT      HARBOR
DOES ROUTING DEPEND ON COLOR    OF SHIP    ?

no

SPECIFY CONDITIONAL (OR "NONE") FOR SHIP

none

NAME NEXT ACTION AND LOCATION FOR SHIP

leave the port

DATA STRUCTURE FOR PORT     SIMULATION IS COMPLETE
```

Figure 8. (CONTINUED)

THE SHIPS ARRIVE AT THE PORT. THE TIME BETWEEN ARRIVALS IS UNIFORMLY DISTRIBUTED, WITH A MEAN OF 30 MINUTES AND A HALF-RANGE OF 60 MINUTES. 20 PERCENT OF THE SHIPS ARE GREEN. 30 PERCENT ARE RED, AND THE REST ARE BLUE. AFTER ARRIVING AT THE PORT, THE SHIPS UNLOAD CARGO AT THE DOCK IN THE PORT. THERE ARE 3 DOCKS IN THE PORT. THE TIME FOR THE SHIPS TO UNLOAD CARGO AT THE DOCK IN THE PORT IS NORMALLY DISTRIBUTED, WITH A MEAN OF 180 MINUTES FOR THE GREEN SHIPS, 240 MINUTES FOR THE RED SHIPS, AND 300 MINUTES FOR THE BLUE SHIPS AND A STANDARD DEVIATION OF 30 MINUTES FOR THE GREEN SHIPS, 60 MINUTES FOR THE RED SHIPS, AND 90 MINUTES FOR THE BLUE SHIPS. AFTER UNLOADING CARGO AT THE DOCK IN THE PORT, THE GREEN SHIPS UNLOAD CARGO AT THE PIER IN THE PORT, THE RED SHIPS UNLOAD CARGO AT THE DEPOT IN THE PORT, AND THE BLUE SHIPS UNLOAD CARGO AT THE BARGE IN THE PORT. THERE ARE 2 PIERS IN THE PORT. THE TIME FOR THE GREEN SHIPS TO UNLOAD CARGO AT THE PIER IN THE PORT IS NORMALLY DISTRIBUTED, WITH A MEAN OF 60 MINUTES AND A STANDARD DEVIATION OF 15 MINUTES. AFTER UNLOADING CARGO AT THE PIER IN THE PORT, 40 PERCENT OF THE SHIPS LOAD CARGO AT THE DOCK IN THE PORT FOR 120 MINUTES, AND THE REST WAIT IN THE HARBOR IN THE PORT UNTIL THE BARGE IS AVAILABLE. THE CAPACITY OF THE DEPOT IS 4 SHIPS. THE TIME FOR THE RED SHIPS TO UNLOAD CARGO AT THE DEPOT IN THE PORT IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN OF 60 MINUTES. AFTER UNLOADING CARGO AT THE DEPOT IN THE PORT, 40 PERCENT OF THE SHIPS LOAD CARGO AT THE DOCK IN THE PORT FOR 120 MINUTES, AND THE REST WAIT IN THE HARBOR IN THE PORT UNTIL THE BARGE IS AVAILABLE. THE TIME FOR THE BLUE SHIPS TO UNLOAD CARGO AT THE BARGE IN THE PORT IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN OF 90 MINUTES. AFTER UNLOADING CARGO AT THE BARGE IN THE PORT, 40 PERCENT OF THE SHIPS LOAD CARGO AT THE DOCK IN THE PORT FOR 120 MINUTES, AND THE REST WAIT IN THE HARBOR IN THE PORT UNTIL THE BARGE IS AVAILABLE. AFTER LOADING CARGO AT THE DOCK IN THE PORT, THE SHIPS WAIT IN THE HARBOR IN THE PORT UNTIL THE BARGE IS AVAILABLE. AFTER WAITING IN THE HARBOR IN THE PORT, THE SHIPS LEAVE THE PORT. THE SIMULATION IS TO BE RUN FOR 5760 MINUTES.

FIGURE 9. ENCODED ENGLISH TEXT OF PORT FACILITY PROBLEM

```
          SIMULATE
          RMULT       277,423,715,121,655,531,999,813
 2        STORAGE     1000
 4        STORAGE     3
 6        STORAGE     2
 8        STORAGE     4
 11       STORAGE     1000
 1        TABLE       M1,0,5,100
 1        FUNCTION    RN1,C24
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.39/
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8/
 2        FUNCTION    RN2,C29
0,-3/.012,-2.25/.027,-1.93/.043,-1.72/.062,-1.54/.084,-1.38/
.104,-1.26/.131,-1.12/.159,-1/.187,-.89/.23,-.74/.267,-.62/.334,-.43/
.432,-.17/.5,0/.568,.17/.666,.43/.732,.62/.77,.74/.813,.89/.841,1/
.869,1.12/.896,1.26/.916,1.38/.938,1.54/.957,1.72/.973,1.93/
.988,2.25/1,3/
 3        FUNCTION    P1,D3
5,180/7,240/9,300/
 4        FUNCTION    RN3,D3
.2,5/.5,7/1,9/
 5        FUNCTION    P1,D3
5,30/7,60/9,90/
 6        FUNCTION    P1,D3
5,LBL3/7,LBL4/9,LBL5/
 7        FUNCTION    RN4,D2
.4,LBL6/1.,LBL7/
          GENERATE    300,60
          ASSIGN      1,FN4
          ENTER       2
 LBL2     QUEUE       4
          ENTER       4
          DEPART      4
          ADVANCE     V1
 1        FVARIABLE   FN3+(FN5*FN2)
          LEAVE       4
          TRANSFER    ,FN6
 LBL3     QUEUE       6
          ENTER       6
          DEPART      6
          ADVANCE     V2
 2        FVARIABLE   60+(15*FN2)
          LEAVE       6
```

FIGURE 10. GPSS REPRESENTATION OF PORT FACILITY PROBLEM

```
        TRANSFER    ,FN7
LBL4    QUEUE       8
        ENTER       8
        DEPART      8
        ADVANCE     60,FN1
        LEAVE       8
        TRANSFER    ,FN7
LBL5    QUEUE       10
        SEIZE       10
        DEPART      10
        ADVANCE     90,FN1
        RELEASE     10
        TRANSFER    ,FN7
LBL6    QUEUE       4
        ENTER       4
        DEPART      4
        ADVANCE     120
        LEAVE       4
LBL7    QUEUE       11
        ENTER       11
        DEPART      11
        GATE NU     10
        LEAVE       11
LBL8    TABULATE    1
        LEAVE       2
        TERMINATE
        GENERATE    5760
        TERMINATE   1
        START       1
        END
```

FIGURE 10. (CONTINUED)

VI. CONCLUSIONS AND RECOMMENDATIONS

The question-answer decoding system provides a convenient method for expressing simple queuing problems and producing the corresponding data structures that enable GES or XGES to encode a GPSS simulation program. The user of this system should know the general concepts of computer simulation, including how to specify and apply probability distributions, and how to run a GPSS program and a analyze the results it produces. A knowledge of GPSS programming is helpful but not essential. The question-answer decoding system and the encoding systems described in Refs. 5 and 6 demonstrate the tremendous potential of NLP for language processing.

A. GENERAL RESULTS

During the development of the question-answer decoding system, a number of techniques were devised which should be useful in future work with NLP. Among them are:

(1) A scheme for collecting, processing and classifying words (usually names) that have not been previously defined.

(2) A convenient method of communicating information from the data structure to the user while in the decoding mode of operation.

(3) The "action-list/successor-attribute" method of developing the action sequence.

(4) The "temporary action" scheme to represent actions that are imminent but must be defined or identified on later cycles.

(5) The overall logic for simulation problem definition that allows an encodable IDS to be generated.

B. FUTURE DEVELOPMENT

There is considerable potential for expansion and future development of the question-answer decoding system. Three areas in which work could be done immediately are:

(1) Expansion of system capability to include such things as shortest line choices, queue length or other condition tests, user specification of GPSS output statistics, and increased flexibility in user input choices.

(2) Development of encoding rules to produce English sentences at the terminal during the question-answer session.

(3) Inclusion of the capability to back up and modify something previously completed, such as an action record, without having to re-construct the entire problem.

APPENDIX A

NLP DECODING RULE SYMBOLOGY AND IMPLEMENTATION

Appendix B of Ref.5 describes NLP encoding rule symbology and usage. The items listed therein which apply to NLP decoding rules have been duplicated as numbers 7 through 28 below. Other items below pertain specifically to the NLP decoding process.

(1) The decoding rules are automatically initiated with the processing of a period (".") followed by a blank (" ") immediately ahead of the terminal input string.

(2) The first character is then read from the input string, and all rules that begin with or are now waiting (as a result of the period and blank) for that character (as a condition on the left) become or remain "active."

(3) Any active rule for which all conditions on the left have been tested is said to be "satisfied," and the SEGMENT's indicated on the right are created. Each newly created SEGMENT is treated in a manner similar, with respect to rule processing, to a character from the input string, and can cause additional rules to become active or be satisfied. This process continues until all possible SEGMENT's have been created and no further rule testing is applicable. Next, all rules which were waiting for a character other than the last input character become inactive, and the next character of the input string is read.

(4) When the input string is exhausted, or (in the case of the question-answer decoding system) a REPLY segment is created, the decoding process is terminated.

(5) When decoding is terminated, all rules become inactive. Also, all SEGMENT records that have not become part of the IDS are erased and their component cells are returned to a free cell list for future use.

(6) When a particular rule is satisfied, the SEGMENT or SEGMENT's that appear on the left are <u>not</u> lost (as in the encoding process) and remain available for other rules until decoding is complete. Thus the concept of "parallel" processing is introduced.

(7) An individual rule is processed from left to right. The portion of the rule to the left of the transformation symbol (-->) is called the "condition" of the rule and is used to test for the applicability of the rule. The portion of the rule to the right of the transformation symbol is called "labeling" and is used to designate the method and extent of new record construction.

(8) A record is explicitly referenced by designating the record name (i.e. SEGMENT TYPE), or, if the record to be referenced is the one currently being tested or constructed, by designating the record name RECORD, SEGMENT, or SEG.

(9) If the record to be referenced is the one currently being tested or constructed, it may be implicitly designated by default.

(10) An attribute may be referenced either by name or by number. The number designation range of legal values is from 11 to 300 and is specified by preceeding the number with the symbol "@".

(11) MEMORY is a unique record in that no copies are made of it. All modifications of MEMORY are made on the original.

(12) Since the rules seldom process original records, the primary way to add, delete, or change attribute information in an original record is to do so via a MEMORY designation.

(13) If a record to be constructed is to be of the same SEGMENT TYPE as the condition record, is to be a copy of the condition record, and only one such record is to be constructed, then only the SEGMENT TYPE need be listed to accomplish the construction.

(14) --> is called the transformation symbol and means "construct the designated records which follow."

(15) (means "of", e.g. SUCC (ACT) mean "successor of action."

(16)) and , are used to assist in reading the rules.

(17) $Q means "test attribute number Q through successive records possessing attribute Q where Q is an integer and attribute Q points to a record that may possess attribute Q. Non-designation of Q will default to 1, signifying the SUP attribute.

(18) .EQ., .NE., .GT., .LT., LE., and .GE. have the same meaning as in FORTRAN and are used for the testing of rule conditions.

(19) = has the same meaning as in FORTRAN.

(20) ¬ means "not" and is used in the testing of rule conditions.

(21) - if not used in an arithmetic expression, means "erase the designated attribute", or turn off the designated indicator.

(22) +, -, *, and / have the standard arithmetic operator meanings. Note that they must be used with attributes of TYPE 0 cell construction and that no more than one operation may be used in an arithmetic expression.

(23) 'XXXX' when appearing to the left of the transformation symbol, means "test to determine if the first SUP attribute of this record points to XXXX."

(24) 'XXXX' when appearing to the right of the transformation symbol, means "assign a pointer to XXXX as the value of the first SUP attribute of this record."

(25) "ZZZ" denotes the character (EBCDIC) representation ZZZ. These double quotes are usually used with an attribute of TYPE 1 cell construction.

(26) | means "or" and applies only to complete test conditions. Also, this symbol must be used for the rightmost test conditions as the rule condition is satisfied whenever any one of the test conditions separated by 1 is satisfied, e.g. BLOK(SUCC.EG.AGENT,GOAL.EQ.AGENT | 'TRANSFER').

(27) @Q means "attribute number Q" where Q is either an integer or an attribute of TYPE 0 cell construction.

(28) % means "make a copy of the designated record to become the structure of this record."

(29) # represents blank space in the input string.

APPENDIX B

ATTRIBUTES, ROUTINES, INDICATORS, AND NAMED RECORDS FOR RULES

ATTRIBUTES:

```
SUP  1, AWORD  8, NUM  9, ANMS  10, JNUM  11, X1  11, X2  12,
X3  13, X4  14, X5  15, X6  16, CLASATR  20, ATRNAM  18,  ERR  29,
A1  31, A2  32, A3  33, A4  34, A5  35, A6  36, A7  37, A8  38,
STORNUM  40, NUMBNAME  41, CHKPOINT  42, CONDTYP  43, IDNAME  44,
SEQNUM  45
```

ROUTINES:

```
RERR  11, RSUP  12, RCHKJ  13, RSETSUP  14, RDONE  15, RGETJ  16,
RCKEND  17, RPRMT1  18, RPRMT2  19, RPRMT3  20, ROK  21, RSTRUC  22,
RMEAN  23, RACTION  24, RPRBTIM  25, RENDPGM  26, RSTOP  27
```

INDICATORS:

```
EXPUSED      1
NORMUSED     2
DOLLARSIGN   3
ADJSW        4
VERBSW       5
DONESW       6
CONTSW       7
OPTNSW       8
PREPSW       9
DISTSW      10
NOUNSW      11
LEAVSW      12
TRANSW      13
USEDSW      14
```

NAMED RECORDS:

```
PROBTIME  ('ATTR')
RNNO  ('ATTR')
TEMP  ('ATTR')
MEPTR  ('ATTR')
SEPTR  ('ATTR')
ACPTR  ('ATTR')
DISTPTR  ('ATTR')
SUCPTR  ('ATTR')
MFNID  ('ATTR')
MVARID  ('ATTR')
```

```
LISTCNTR  ('ATTR')
IDNO  ('ATTR')
LOCATION  ('ATTR',NOUNSW)
PRED  ('ATTR')
SUCC  ('ATTR')
AGENT  ('ATTR')
GOAL  ('ATTR')
DURATION  ('ATTR')
IETM  ('ATTR')
ASNDISTR  ('ATTR')
QUANTITY  ('ATTR')
SIZE  ('ATTR',OPTNSW,TYP='RELSIZ')
COLOR  ('ATTR',OPTNSW,TYP='ABSCOLR')
WEIGHT  ('ATTR',OPTNSW,TYP='ABSWEIT')
MBNTY  ('ATTR')
STORIND  ('ATTR')
IDENT  ('ATTR')
STRUC  ('ATTR')
CONSUMP  ('ATTR')
CAPACITY  ('ATTR')
MEAN  ('ATTR')
RANGE  ('ATTR')
STDEV  ('ATTR')
FNID  ('ATTR')
FNARG  ('ATTR')
DORC  ('ATTR')
XYLAST  ('ATTR')
PNUM  ('ATTR')
SUCARG  ('ATTR')
MAXQ  ('ATTR')
OPENACT  ('ATTR')
CLOSACT  ('ATTR')
ARGA  ('ATTR')
ARGB  ('ATTR')
LABL  ('ATTR')
BLOKMOD  ('ATTR')
MODREL  ('ATTR')
OBJREL  ('ATTR')
LOCOBJ  ('ATTR')
LASTREC  ('ATTR')
CHARS  ('ATTR')
CONDENTY  ('ATTR')
ARGAZ  ('ATTR')
ADJP  ('ATTR')
ADJECT  ('ATTR')
FLAG  ('ATTR')
UNKNOWN('ATTR');
ARRIV  ('EVENT')
```

```
LEAV  ('EVENT')
GOTO        ('EVENT')
START('EVENT')
WAIT  ('ACTIVITY')
UNLOAD  ('ACTIVITY')
LOAD  ('ACTIVITY')
SERVIC  ('ACTIVITY')
EVENT  ('ACTION')
ACTIVITY  ('ACTION')
SHIP  ('MOBENTY')
CUSTOMER    ('MOBENTY')
CAR  ('MOBENTY')
HARBOR  ('STATENTY')
PIER  ('STATENTY')
DOCK  ('STATENTY')
PORT  ('STATENTY')
DEPOT  ('STATENTY')
BARGE  ('STATENTY')
CARGO  ('STATENTY')
BANK        ('STATENTY')
WINDOW      ('STATENTY')
GASSTA  ('STATENTY')
MOBLIST  ('RECLIST')
STALIST  ('RECLIST')
DSTRLIST  ('RECLIST')
SCSRLIST  ('RECLIST')
ACTNLIST  ('RECLIST')
PREDLIST  ('RECLIST')
SELARGS     ('RECLIST')
MOBENTY  ('ENTITY')
STATENTY  ('ENTITY')
OBJENTY('ENTITY')
EMPDIST  ('DISTR1')
TYPDIST  ('DISTR1')
UNIFORM  ('DISTR2')
EXPON  ('DISTR2')
NORMAL  ('DISTR2')
TYPTABL  ('TABL')
TABL  ('FNCTN')
DISTR1  ('FNCTN')
COND1  ('COND')
COND2  ('COND')
IN  ('LOCDESCR',PREPSW)
ON  ('LOCDESCR',PREPSW)
NEAR  ('LOCDESCR',PREPSW)
AT  ('LOCDESCR',PREPSW)
AROUND  ('LOCDESCR',PREPSW)
TO  (PREPSW)
```

```
FRACTNL  ('SUCDSCR1')
PTYP  ('SUCDSCR1')
QTYP  ('SUCDSCR2')
STYP  ('SUCDSCR2')
FTYP  ('SUCDSCR2')
SUCDSCR1  ('SUCDESCR')
SUCDSCR2  ('SUCDESCR')
IS ('ASGNMT',VERBSW)
EQUAL ('ASGNMT',VERBSW)
ARE ('ASGNMT',VERBSW)
LOCDESCR ('DESCR')
ASGNMT ('DESCR')
GT  ('RELIND1')
NG  ('RELIND1')
EQ  ('RELIND1')
NE  ('RELIND1')
LT  ('RELIND1')
NL  ('RELIND1')
ER  ('RELIND2')
EST  ('RELIND2')
RELIND1  ('RELTV')
RELIND2  ('RELTV')
FAST  ('RELTIME')
SLOW  ('RELTIME')
MANY  ('RELQNTY')
FEW  ('RELQNTY')
DARK  ('RELCOLR',ADJSW)
LIGHT1  ('RELCOLR')
HEAVY  ('RELWEIT',ADJSW)
LIGHT2  ('RELWEIT')
LARGE  ('RELSIZ')
SMALL  ('RELSIZ')
BIG  ('RELSIZ')
LITTLE  ('RELSIZ')
RELTIME  ('RELVAL')
RELQNTY  ('RELVAL')
RELCOLR  ('RELVAL')
RELWEIT  ('RELVAL')
RELSIZ  ('RELVAL')
HOUR  ('ABSTIME')
MINUTE  ('ABSTIME')
SECOND  ('ABSTIME')
TON  ('ABSWEIT')
POUND  ('ABSWEIT')
OUNCE  ('ABSWEIT')
DECIMAL  ('ABSQNTY')
UNIT  ('ABSQNTY')
PERCENT  ('ABSQNTY')
```

```
RED   ('ABSCOLR')
ORANGE   ('ABSCOLR')
YELLOW   ('ABSCOLR')
GREEN   ('ABSCOLR')
BLUE   ('ABSCOLR')
VIOLET   ('ABSCOLR')
BLACK   ('ABSCOLR')
WHITE   ('ABSCOLR')
ABSTIME   ('QUANVAL')
ABSWEIT   ('QUANVAL')
ABSQNTY   ('QUANVAL')
NUMBER   ('QUANVAL')
ABSCOLR   ('QUALVAL')
NAME ('QUALVAL')
VARIABLE ('ARGNAME')
PARAMETR ('ARGNAME')


FUNCTION ('ARGNAME')
PARAMNO   ('ARGVAL')
FUNCNO   ('ARGVAL')
RANDM   ('ARGVAL')
RELVAL   ('VAL')
QUANVAL   ('VAL')
QUALVAL   ('VAL')

MOBL ('MOBLIST',LASTREC=10)
STAL ('STALIST', LASTREC=10)
ACTL ('ACTNLIST', LASTREC=10)
DSTL ('DSTRLIST', LASTREC=10)
SCRL ('SCSRLIST', LASTREC=10)
CNDL ('CONDTNLS', LASTREC=10)
SETMEM (RNNO(MEM)=1,IDACT(MEM)=1, IDETY(MEM)=1, MEPTR(MEM)='MOBL',
        SUCPTR(MEM)='SCRL',DISTPTR(MEM)='DSTL',MVARID(MEM)=1,
        SEPTR(MEM)='STAL', ACPTR(MEM)='ACTL', TYPENT(MEM)='START',
        MFNID(MEM)=3, PNUM(MEM)=1)

A          (ADJSW)
AN         (ADJSW)
THE        (ADJSW)
LIGHT      (ADJSW)
OPTIONAL   (ADJSW)
IDENTICA   (ADJSW)
AM         (VERBSW)
HAVE       (VERBSW)
WAIT       (VERBSW)
DELAY      (VERBSW)
LOCATED    (VERBSW)
```

```
ATTRIBUT   (NOUNSW)
EXIT       (LEAVSW)
LEAVE      (LEAVSW)
DEPART     (LEAVSW)
N          (DONESW)
NO         (DONESW)
END        (DONESW)
DONE       (DONESW)
NONE       (DONESW)
FINISH     (DONESW)
FINISHED   (DONESW)
Y          (CONTSW)
YES        (CONTSW)
MORE       (CONTSW)
CONTINUE   (CONTSW)
TRANS      (TRANSW)
TRANSFER   (TRANSW)
EXP        (DISTSW,TYP=1)
EXPON      (DISTSW,TYP=1)
EXPONENT   (DISTSW,TYP=2)
CONSTANT   (DISTSW,TYP=2)
NORM       (DISTSW,TYP=3)
NORMAL     (DISTSW,TYP=3)
UNIF       (DISTSW,TYP=4)
UNIFORM    (DISTSW,TYP=4)
```

APPENDIX C

QUESTION-ANSWER SYSTEM DECODING RULES

LEXOLOGY FOR QUESTIONAIRE DECODING:

```
. /# --> PERIOD
.    -->   PUNCA, PUNCB, PUNCC
#    -->   PUNCB, PUNCC, PUNCD
,    -->   PUNCA, PUNDB, PUNCD
=    -->   PUNCB, PUNCD, WORD('EQUAL')
PUNCD #  -->  PUNCD
PUNCD ,  -->  PUNCD

A --> LETTER*
B --> LETTER
C --> LETTER
D --> LETTER
E --> LETTER
F --> LETTER
G --> LETTER
H --> LETTER
I --> LETTER
J --> LETTER
K --> LETTER
L --> LETTER
M --> LETTER
N --> LETTER
O --> LETTER
P --> LETTER
Q --> LETTER
R --> LETTER
S --> LETTER
T --> LETTER
U --> LETTER
V --> LETTER
W --> LETTER
X --> LETTER
Y --> LETTER
Z --> LETTER
PUNCD LETTER   -->  WORDP(RGETJ)
WORDP DIGIT    -->  WORDP
WORDP LETTER   -->  WORDP
WORDP /PUNCB   -->  WORD(JNUM(WORDP),RSETSUP)
./ WORD(RCHKJ)  -->  WORDSEG(%WORD)
```

```
WORD(PREPSW*)   -->  PREP(SUP(WORD))
WORD(NOUNSW*)   -->  NOUN(SUP(WORD))
WORD(ADJSW*)    -->  ADJ(SUP(WORD))
WORD(DONESW*)   -->  DONEP(SUP(WORD))
WORD(CONTSW*)   -->  CONTINUP(SUP(WORD))
WORD(LEAVSW*)   -->  LEAVS(SUP(WORD))
WORD(VERBSW*)   -->  VERB(SUP(WORD))
VERB('DELAY')   -->  VERB('WAIT')
WORD(DISTSW*)   -->  DISTSEG1(SUP(WORD),X5(MEM)=TYP(SUP))
WORD(OPTNSW*)   -->  OPATRS(ATTRB=SUP(WORD),SUP=TYP(ATTRB))
WORD(TRANSW*,CHKPOINT(MEM).EQ.5)  -->  FRDIST('FRACTNL')
WORD('STOPIT')  -->  REPLY(-CHKPOINT(MEM),RSTOP,-CURSEG(MEM))

0  -->  DIGIT(NUM=0)
1  -->  DIGIT(NUM=1)
2  -->  DIGIT(NUM=2)
3  -->  DIGIT(NUM=3)
4  -->  DIGIT(NUM=4)
5  -->  DIGIT(NUM=5)
6  -->  DIGIT(NUM=6)
7  -->  DIGIT(NUM=7)
8  -->  DIGIT(NUM=8)
9  -->  DIGIT(NUM=9)
# DIGIT  -->  NUMBERP(%DIGIT)
NUMBERP DIGIT  -->  NUMBERP(NUM=10*NUM, NUM=NUM+NUM(DIGIT))
NUMBERP /PUNCB  -->  NMBER(%NUMBERP, SUP='UNIT')

:  -->  SPECHAR('COLON')
;  -->  SPECHAR('SEMICOLN')
'  -->  SPECHAR('APOSTPHE')
"  -->  SPECHAR('QUOTE')
$  -->  SPECHAR('DOLRSIGN')
?  -->  SPECHAR('QUESTION')

NOUN S  -->  NOUN

DONEP('NO') CONTINUP('MORE')  -->  DONEP('MOMORE')
DONEP('NOMORE') WORD('MOBILE')    -->  DONEP
DONEP('NOMORE') WORD('STATIONA')  -->  DONEP
DONEP('NOMORE') WORD('ENTITIES')  -->  DONEP
DONEP('NOMORE') ADJ('OPTIONAL') NOUN('ATTRIBUT') --> DONEP
./ DONEP(RCHKJ)  -->  CKDONE(SUP(CURSEG(MEM)))
CKDONE(¬CURSEG(MEM),¬CHKPOINT(MEM)) -->  REPLYP(RDONE)

./ CONTINUP(RCHKJ)  -->  CONTNUSG(SUP(CURSEG(MEM)))
CONTNUSG(SUP.EQ.0)  -->  REPLY1
```

```
    REPLYCHK -->  REPLYP(-CHKPOINT(MEM),-CURSEG(MEM))
    REPLYP  -->  REPLY1(RCKEND, ERR(MEM)=2,RERR)
    REPLY1  -->  REPLY(ROK)
    REPLY  -->  NOTHING

SEARCH(PREVREC.EQ.0) --> SEARCH2(LIST(MEM)=LIST(SEARCH),CODE(MEM)=
                                              CODE(SEARCH),LC(MEM)=10)
SEARCH(PREVREC.NE.0) --> SEARCH1(%SEARCH,LIST(MEM)=LIST,CODE(MEM)=CODE,
                                                        LC(MEM)=11)

SEARCH1(PREVREC.EQ.@LC(MEM)(LIST))  -->  SEARCH2
SEARCH1(PREVREC.NE.@LC(MEM)(LIST))  -->  SEARCH1(LC(MEM)=LC(MEM)+1)
SEARCH2  -->  SEARCH3(LC(MEM)=LC(MEM)+1,REC(MEM)=@LC(MEM)(LIST(MEM)))
SEARCH3(REC(MEM).EQ.0) -->  SRCHFAIL(CODE(MEM))
SEARCH3(REC(MEM).NE.0) -->  SRCHREC*1(%REC(MEM),CODE(MEM))

NMBER(STORNUM(MEM),NUMBNAME(MEM)) --> REPLYP(ST=STORNUM(MEM),
          SUP(NMBER)=NUMBNAME(MEM),@ST(SEG)(CURSEG(MEM))=NMBER,
          -STORNUM(MEM),-NUMBNAME(MEM),ST.EQ.'CAPACITY',
                                  OPTATTR(CURSEG(MEM))=1)

NMBER(STORNUM(MEM),¬NUMBNAME(MEM)) -->
   REPLYP(@STORNUM(MEM)(CURSEG(MEM))=NUM(NMBER),-STORNUM(MEM))

OPATRS PUNCD  -->  OPATRS
OPATRS PREP('OF') ADJ('THE')  -->  OPATRS
OPATRS WORD($'ENTITY')  -->  OPATRS
OPATRS WORD($'DESCR')  -->  OPATRS
OPATRS =  -->  OPATRS
OPATRS NMBER  -->  OPATRS(NUM(NMBER))

OPATRS(OPTATTR(CURSEG(MEM)),'ABSWEIT',¬NUM) ADJ($'RELWEIT'|'LIGHT') -->
          REPLYP(WEIGHT(CURSEG(MEM))=SUP(ADJ),SUP(SUP(ADJ))='RELWEIT')

OPATRS(OPTATTR(CURSEG(MEM)),'ABSCOLR',¬NUM) ADJ($'RELCOLR'|'LIGHT') -->
        REPLYP(SUP(SUP(ADJ))='RELCOLR',COLOR(CURSEG(MEM))=SUP(ADJ))

OPATRS(OPTATTR(CURSEG(MEM)),¬NUM) WORD($SUP(OPATRS)|$'UNKNOWN')  -->
   REPLYP(%OPATRS,AWORD=SUP(WORD),RSUP,@ATTRB(SEG)(CURSEG(MEM))=SUP)

OPATRS(OPTATTR(CURSEG(MEM)),NUM) WORD($'VAL'|$'UNKNOWN') -->
   REPLYP(%OPATRS,AWORD=SUP(WORD),RSUP,@ATTRB(SEG)(CURSEG(MEM))=SEG,
                          -ATTRB,-AWORD)

WORD(OPTATTR(CURSEG(MEM)),$'UNKNOWN') --> ATRPRT1(ATTRB=SUP(WORD),
                                          SUP(ATTRB)='ATTR')

ADJ('THE') ATRPRT1  -->  ATRPRT1
```

```
ATRPRT1 PREP('OF') ADJ('THE') WORD($'ENTITY')  -->  ATRPRT1

./ ATRPRT1 PREP($'DESCR') --> ATTRPART(ATTRB(ATRPRT1))

./ ATRPRT1 VERB($'DESCR') --> ATTRPART(ATTRB(ATRPRT1))

ATTRPART WORD(RCHKJ) --> REPLY(@ATTRB(ATTRPART)(CURSEG(MEM))=SUP(WORD),
          SUP(SUP(WORD)).EQ.'UNKNOWN',SUP(SUP(WORD))='QUALVAL')

ATTRPART NMBER(RCHKJ) --> REPLY(@ATTRB(ATTRPART)(CURSEG(MEM))=NUM(NMBER))

VERB('LOCATED') PREP($'LOCDESCR') --> LOCATNP(SUP(PREP))

NOUN('LOCATION') VERB($'ASGNMT') --> NOUN('LOCATION')

NOUN('LOCATION') PREP($'LOCDESCR') --> LOCATNP(SUP(PREP))

NOUN('LOCATION') /WORD($'UNKNOWN'|$'STATENTY') --> LOCATNP('AT')

LOCATNP ADJ('THE')  --> LOCATNP

LOCATNP WORD(¬$'UNKNOWN',¬$'STATENTY') --> REPLYP(SUP(WORD),
                                        ERR(MEM)=4,RERR)

LOCATNP WORD($'UNKNOWN') --> REPLY1(SUP(WORD),SUP(SUP)='STATENTY',
      LOCOBJ(LOCATNP)=SEG,LOCATION(CURSEG(MEM))=LOCATNP,QUANTITY=1,
      STORIND=1,IDNO=IDETY(MEM),IDETY(MEM)=IDETY(MEM)+1,
      LASTREC('STAL')=LASTREC('STAL')+1,@LASTREC('STAL')('STAL')=SEG)

LOCATNP WORD($'STATENTY') --> SEARCH(LIST='STAL',CODE=10,SUP(MEM)=SUP(WORD))

SRCHREC(CODE.EQ.10,SUP=SUP(MEM)) --> REPLYP(LOCATION(CURSEG(MEM))=REC(MEM))

SRCHFAIL(CODE.EQ.10) --> REPLY1(SUP(MEM),ERR(MEM)=5,RERR)

WORD(CHKPOINT(MEM).EQ.3,¬$'ABSTIME',¬$'UNKNOWN')  --> REPLY(SUP(WORD),
                                        ERR(MEM)=17,RERR)

WORD(CHKPOINT(MEM).EQ.3,$'ABSTIME'|$'UNKNOWN') --> REPLY1(-CHKPOINT(MEM),
          TIMEUNIT(MEM)=SUP(WORD), SUP(SUP(WORD))='ABSTIME')

WORDSEG(TYPENT(MEM).EQ.'START',$'STATENTY'|$'UNKNOWN') -->
      REPLYCHK(SUP(WORDSEG),SUP(SUP)='STATENTY',IDNO=1,IDETY(MEM)=2,
      QUANTITY=1,STORIND=1,@11('STAL')=SEG,LASTREC('STAL')=11,
                                        TYPENT(MEM)='STATENTY')

WORDSEG(TYPENT(MEM).EQ.'STATENTY',¬CURSEG(MEM),$'STATENTY'|$'UNKNOWN') -->
   REPLY1(SUP(WORDSEG),A1(MEM)=ANMS(SUP),SUP(SUP)='STATENTY',STORIND=1,
```

```
        IDNO=IDETY(MEM),IDETY(MEM)=IDETY(MEM)+1,
           CURSEG(MEM)=SEG,-CHKPOINT(MEM))

WORDSEG(¬$'STATENTY',¬$'UNKNOWN',¬CURSEG(MEM),TYPENT(MEM).EQ.'START'|
     TYPENT(MEM).EQ.'STATENTY') --> REPLY(ERR(MEM)=13,SUP(WORDSEG),RERR)

CKDONE($'STATENTY',OPTATTR(CURSEG(MEM))) --> REPLYCHK(RDONE,
      -OPTATTR(CURSEG(MEM)),LASTREC('STAL')=LASTREC('STAL')+1,
      @LASTREC('STAL')('STAL')=CURSEG(MEM),
      NUM(@CAPACITY(CURSEG(MEM))).EQ.1,-STORIND(CURSEG(MEM)))

CKDONE(CHKPOINT(MEM).EQ.1) --> REPLYCHK(TYPENT(MEM)='MOBENTY',RDONE)

WORDSEG(TYPENT(MEM).EQ.'MOBENTY',¬CURSEG(MEM),$'MOBENTY'|
      $'UNKNOWN') --> REPLY1(SUP(WORDSEG),SUP(SUP)='MOBENTY',
                        CURSEG(MEM)=SEG,A1(MEM)=ANMS(SUP),-CHKPOINT(MEM))

WORDSEG(TYPENT(MEM).EQ.'MOBENTY',¬CURSEG(MEM),¬$'MOBENTY',
      ¬$'UNKNOWN') --> REPLY(ERR(MEM)=16,SUP(WORDSEG),RERR)

CKDONE($'MOBENTY',¬IDNO(CURSEG(MEM)),TYPENT(MEM).EQ.'MOBENTY') -->
     REPLY1(IDNO(CURSEG(MEM))=IDETY(MEM),IDETY(MEM)=IDETY(MEM)+1,
       OPTATTR(CURSEG(MEM))=1,RDONE)

CONTNUSG($'MOBENTY',¬IDNO(CURSEG(MEM)),TYPENT(MEM).EQ.'MOBENTY') -->
   REPLYP(IDNO(CURSEG(MEM))=IDETY(MEM),IDETY(MEM)=IDETY(MEM)+1,
                        CLASATR(CURSEG(MEM))=1)

WORDSEG(CURSEG(MEM)$'MOBENTY',CLASATR(CURSEG(MEM)).EQ.1,
      $'ATTR'|$'UNKNOWN') --> REPLY1(CLASATR(CURSEG(MEM))=SUP(WORDSEG),
      SUP(SUP(WORDSEG))='ATTR',LASTREC('MOBL')=LASTREC('MOBL')+1,
      @LASTREC('MOBL')('MOBL')=CURSEG(MEM),SUP(CURSEG(MEM)),OPTATTR=1,
      STRUC=CURSEG(MEM),CURSEG(MEM)=SEG,A1(MEM)=ANMS(SUP(WORDSEG)))

WORDSEG(CURSEG(MEM)$'MOBENTY',CLASATR(CURSEG(MEM)).EQ.1,¬$'ATTR',
      ¬$'UNKNOWN') --> REPLY(ERR(MEM)=6,SUP(WORDSEG),RERR)

CKDONE($'MOBENTY',¬STRUC(CURSEG(MEM)),OPTATTR(CURSEG(MEM))) -->
      REPLYCHK(-OPTATTR(CURSEG(MEM)),LSATREC('MOBL')=LASTREC('MOBL')+1,
            @LASTREC('MOBL')('MOBL')=CURSEG(MEM),RDONE)

CKDONE($'MOBENTY',STRUC(CURSEG(MEM)),OPTATTR(CURSEG(MEM))) -->
   PNUMREC('UNIT',NUM=PNUM(MEM),PNUM(MEM)=PNUM(MEM)+1,
         TEMPSEG(MEM)=SEG,-OPTATTR(CURSEG(MEM))),
   DISTREC('MOBTEMP',FNID=MFNID(MEM),MFNID(MEM)=MFNID(MEM)+1,
         XYLAST=100,PNUM=TEMPSEG(MEM),PERCENT(MEM)=0,RPRMT1)

DISTREC(PERCENT(MEM).LT.99,'MOBTEMP') --> PCNTREC('PERCENT',RPRMT2,
```

```
          NUM=X3(MEM),PERCENT(MEM)=PERCENT(MEM)+NUM,TEMPSEG(MEM)=SEG),
   MOBENY(%CURSEG(MEM),IDNO=IDETY(MEM),IDETY(MEM)=IDETY(MEM)+1,
          @CLASATR(STRUC)(SEG)=X2(MEM),QUANTITY=TEMPSEG(MEM),
          LASTREC('MOBL')=LASTREC('MOBL')+1,
                          @LASTREC('MOBL')('MOBL')=SEG),
   DECMREC('DECIMAL',TEMPSEG(MEM)=SEG,NUM=PERCENT(MEM)*10,
                                NUM.GE.990,NUM=1000),
   DISTREC(XYLAST=XYLAST+1,@XYLAST(SEG)=TEMPSEG(MEM),XYLAST=
          XYLAST=XYLAST+1,@XYLAST(SEG)=@LASTREC('MOBL')('MOBL'))

DISTREC(PERCENT(MEM).GE.99,'MOBTEMP') --> TMPS('RANDM',TEMPSEG(MEM)=SEG),
     REPLY1(%DISTREC,'TYPDIST',DORC="D",FNARG=TEMPSEG(MEM),
          LASTREC('DSTL')=LASREC('DSTL')+1,
          ASNDISTR(STRUC(@LASTREC('MOBL')('MOBL')))=DISTREC,
          @LASTREC('DSTL')('DSTL')=SEG,-CURSEG(MEM),RDONE)

CKDONE(CHKPOINT(MEM).EQ.2) --> TMPS('AT',LOCOBJ=@11('STAL'),
       TEMPSEG(MEM)=SEG), ARRIVS(LOCATION=TEMPSEG(MEM),
                          CURMOB(MEM)=@11('MOBL'))

ARRIVS --> REPLY1('ARRIV',IDNO=IDACT(MEM),IDACT(MEM)=IDACT(MEM)+1,
       MBNTY=CURMOB(MEM),LOCATION(ARRIVS),GETDIST(MEM)='IETM',
        @ASNDISTR(MBNTY).NE.0,ASNDISTR(MBNTY),
       CURSEG(MEM)=SEG,A1(MEM)=ANMS(SUP(MBNTY)),A2(MEM)=
       ANMS(SUP(LOCOBJ(LOCATION))),TYPENT(MEM)=SUP,RDONE,-CHKPOINT(MEM))

CKDONE(CHKPOINT(MEM).EQ.4) --> REPLYP(RDONE,-CHKPOINT(MEM))

CONTNUSG(CHKPOINT(MEM).EQ.4) --> TMPS('PARAMNO',TEMPSEG(MEM)=SEG,
                                 NUM(PNUM(ASNDISTR(MEM)))),
   SETPTYP('PTYP',SUCARG=TEMPSEG(MEM),XYLAST=100,-CHKPOINT(MEM),
                                 CURSEG(MEM)=SEG),
   SEARCH(LIST='MOBL',PREVREC=CURMOB(MEM),CODE=1)

SRCHREC(CODE.EQ.1,SUP.EQ.SUP(CURMOB(MEM)),@CLASATR(CURMOB(MEM))) -->
   SEARCH2('TEMPACT',POINT=CURSEG(MEM),LASTREC('ACTL')=LASTREC('ACTL')+1,
       @LASTREC('ACTL')('ACTL')=SEG,ALPOS=LASTREC('ACTL'),LAM=LASTACT(MEM),
       MBNTY=REC(MEM),A1=ANMS(SUP(MBNTY)),A2=ANMS(SUP(LAM)),
       A3=ANMS(SUP(LOCATION(LAM))),A4=ANMS(SUP(LOCOBJ(LOCATION(LAM)))),
       A5=" ",A6=" ",CONDTYP=1,XY=XYLAST(CURSEG(MEM))+1,
       @XY(SEG)(CURSEG(MEM))=REC(MEM),XY=XY+1,
       @XY(SEG)(CURSEG(MEM))=SEG,XYLAST(CURSEG(MEM))=XY,-LAM)

SRCHFAIL(CODE.EQ.1) -->  CHKACTN

NEXTACT --> SEARCH(LIST='ACTL',PREVREC=LASTACT(MEM),CODE=2)

SRCHREC(CODE.EQ.2,¬SUCC,¬$'LEAV') --> STARTACT(LASTACT(MEM)=REC(MEM),
```

```
                                                        -X3(MEM))

SRCHFAIL(CODE.EQ.2) --> SEARCH(LIST='MOBL',PREVREC=CURMOB(MEM),CODE=8)

STARTACT(SUP(LASTACT(MEM)).NE.'TEMPACT') --> STRTACT1(LAM=LASTACT(MEM),
       AGT=MBNTY(MEM),CURMOB(MEM)=AGT,X1(MEM)=1,STRUC(AGT),
       A1(MEM)=ANMS(SUP(AGT)),A4(MEM)=ANMS(SUP(LOCOBJ(LOCATION(LAM)))),
       A3(MEM)=ANMS(SUP(LOCATION(LAM))),A2(MEM)=ANMS(SUP(LAM)),
       A5(MEM)=" ",A6(MEM)=" ",X2(MEM)=CLASATR(AGT))

STARTACT(SUP(LASTACT(MEM)).EQ.'TEMPACT') -->  STRTACT1(LAM=LASTACT(MEM),
       AGT=MBNTY(LAM),X1(MEM)=1,CURMOB(MEM)=AGT,A1(MEM)=ANMS(SUP(AGT)),
       A2(MEM)=A2(LAM),A3(MEM)=A3(LAM),A4(MEM)=A4(LAM),A5(MEM)=A5(LAM),
       A6(MEM)=A6(LAM),STRUC(AGT),X2(MEM)=CLASATR(AGT))

STRTACT1(STRUC.NE.0) --> STRTACT1(A6(MEM)=A5(MEM),A5(MEM)=A4(MEM),
       A4(MEM)=A3(MEM),A3(MEM)=A2(MEM),A2(MEM)=A1(MEM),
       A1(MEM)=ANMS(@CLASATR(STRUC)(AGT)),AGT=STRUC,STRUC(AGT),
       X1(MEM)=X1(MEM)+1,X1(MEM).EQ.2,X3(MEM)=@CLASATR(STRUC)(AGT))

STRTACT1(STRUC.EQ.0) --> REPLY1(MBNTY(MEM)=MBNTY(LASTACT(MEM)),
       X6(MEM)=SEQNUM(LASTACT(MEM)), X4(MEM)=CONDTYP(LASTACT(MEM)),
       X5(MEM)=NUM(LASTACT(MEM)))

DISTSEG1(CHKPOINT(MEM).NE.7|¬CLASATR(MBNTY(CURSEG(MEM)))) -->
                                                DISTSEG3(SUP(DISTSEG1))

DISTSEG1(CHKPOINT(MEM).EQ.7,CLASATR(MBNTY(CURSEG(MEM)))) -->
   REPLY1(A2(MEM)=ANMS(CLASATR(MBNTY(CURSEG(MEM)))),REC(MEM)=DISTSEG1)

DISTSEG3 --> TMPS(A1(MEM)=ANMS(SUP(DISTSEG3)),RMEAN,SUP(TIMEUNIT(MEM)),
                        NUM=X3(MEM),TEMPSEG(MEM)=SEG),
   DISTSEG2(SUP(DISTSEG3),MEAN=TEMPSEG(MEM),SUP(SEG).EQ.'CONSTANT',
                                                %MEAN,-MEAN)

DISTSEG2('UNIFORM'|'NORMAL') --> TMPS(SUP=TIMEUNIT(MEM),RPRMT3,NUM=X3(MEM),
                                                TEMPSEG(MEM)=SEG),
   DISTSEG(%DISTSEG2,RANGE=TEMPSEG(MEM),SUP.EQ.'NORMAL',STDEV=RANGE,-RANGE)

DISTSEG2(¬'NORMAL',¬'UNIFORM')  -->  DISTSEG(%DISTSEG2)

DISTSEG(TYPENT(MEM).NE.'ARRIV',CURSEG(MEM)$'ACTIVITY') -->
   NEWACTN(@GETDIST(MEM)(CURSEG(MEM))=DISTSEG,RDONE)

DISTSEG(TYPENT(MEM).NE.'ARRIV',CURSEG(MEM)$'EVENT')  -->
   CHKACTN(@GETDIST(MEM)(CURSEG(MEM))=DISTSEG,RDONE)

DISTSEG(TYPENT(MEM).EQ.'ARRIV') --> STARTACT(TYPENT(MEM)='ACTION',
```

```
          @GETDIST(MEM)(CURSEG(MEM))=DISTSEG,LASTACT(MEM)=CURSEG(MEM))

CKDONE(CHKPOINT(MEM).EQ.5) --> REPLYP(RDONE)

FRDIST PUNCD WORD(¬$'UNKNOWN') --> REPLY(SUP(WORD),ERR(MEM)=17,RERR)

FRDIST PUNCD WORD($'UNKNOWN') --> FRDIST1(%FRDIST,IDNAME=SUP(WORD),
                                          SUP(IDNAME)='NAME')

FRDIST1('FRACTNL') --> SEARCH(LIST='CNDL',CODE=4,CURSEG(MEM)=FRDIST1)

SRCHREC(CODE.EQ.4,SUP.EQ.'FRACTNL',IDNAME=IDNAME(CURSEG(MEM))) -->
              CHKACTN(CURSEG(MEM)=REC(MEM))

SRCHFAIL(CODE.EQ.4) --> TMPS('RANDM',TEMPSEG(MEM)=SEG),
   REPLYP(%CURSEG(MEM),SUCARG=TEMPSEG(MEM),XYLAST=100,SEQNUM(MEM)=1,
       PERCENT(MEM)=0,A1(MEM)=ANMS(IDNAME),CURSEG(MEM)=SEG,X4(MEM)=2,
       REC(MEM)=SEG,MBNTY(LASTACT(MEM)),STRUC(MBNTY).NE.0,A2(MEM)=
       ANMS(SUP(MBNTY)),A3(MEM)=ANMS(@CLASATR(STRUC(MBNTY))(MBNTY)),
                                  RSTRUC,MBNTY=STRUC(MBNTY))

./ NMBER  -->  FRCHK(NUM(NMBER))

FRCHK(CHKPOINT(MEM).EQ.5,SUP(CURSEG(MEM)).EQ.'FRACTNL',PERCENT(MEM).LT.985)
   -->  FRNUM(NUM(FRCHK),PERCENT(MEM)=PERCENT(MEM)+10*NUM)

FRNUM %   -->   FRNUM

FRNUM /PUNCD --> TMPS('DECIMAL',NUM=PERCENT(MEM),TEMPSEG(MEM)=SEG),
   TEMPACTN*1('TEMPACT',POINT=CURSEG(MEM),LASTREC('ACTL')=LASTREC('ACTL')+1,
       @LASTREC('ACTL')=SEG,ALPOS=LASTREC('ACTL'),MBNTY(CURSEG(MEM)),
       A2="DEPARTS",A3="THE EVEN",A4="T NAMED",A5=A1(MEM),A6=" ",
       XY=XYLAST(CURSEG(MEM))+1,@XY(SEG)(CURSEG(MEM))=TEMPSEG(MEM),
       XY=XY+1,@XY(SEG)(CURSEG(MEM))=SEG,XYLAST(CURSEG(MEM))=XY,
       CONDTYP=2,NUM(FRNUM),SEQNUM(MEM),SEQNUM(MEM)=SEQNUM(MEM)+1)

TEMPACTN(SUP(CURSEG(MEM)).EQ.'FRACTNL',PERCENT(MEM).LT.985,RCHKJ) -->
   REPLYCHK(SUP(CURSEG(MEM)),ERR(MEM)=11,RERR)

TEMPACTN(SUP(CURSEG(MEM)).EQ.'FRACTNL',PERCENT(MEM).GE.985) -->
   CHKACTN(XY=XY(TEMPACTN)-1,NUM(@XY(SEG)(CURSEG(MEM)))=1000,
       LASTREC('CNDL')=LASTREC('CNDL')+1,@LASTREC('CNDL')('CNDL')=REC(MEM))

TEMPACTN PUNCD NMBER  -->  FRCHK(NUM(NMBER))

VERB('WAIT', CHKPOINT(MEM).EQ.5) PUNCD WORD($'UNKNOWN')  -->
   REPLY1('WAIT',IDNAME=SUP(WORD),SUP(IDNAME)='NAME',X4(MEM)=3,
                                  CURSEG(MEM)=SEG,A1(MEM)=ANMS(IDNAME))
```

```
VERB('WAIT',CHKPOINT(MEM).EQ.5) PUNCD WORD(¬$'UNKNOWN')  -->
                              REPLYP(ERR(MEM)=17,SUP(WORD),RERR)

./ WORD($'STATENTY',CHKPOINT(MEM).EQ.5,SUP(CURSEG(MEM)).EQ.'WAIT')  -->
       CONDP('AT',LOCOBJ=SUP(WORD),LOCATION(CURSEG(MEM))=SEG)

CONDP PUNCD WORD($'STATENTY') --> CONDP1(CONDENTY=SUP(WORD),
                                         DURATION(CURSEG(MEM))=SEG)

CONDP1 PUNCD N O T # F U L L  -->  CONDP2

CONDP1 PUNCD N O T # I N # U S E  -->  CONDP2

CONDP1 PUNCD F U L L  -->  CONDP3

CONDP1 PUNCD I N # U S E  -->  CONDP3

CONDP2  -->  CONDP4(SUP(DURATION(CURSEG(MEM)))='COND1')

CONDP3  -->  CONDP4(SUP(DURATION(CURSEG(MEM)))='COND2')

CONDP4 --> SEARCH(LIST='CNDL',CODE=3,MBNTY(MEM)=MBNTY(LASTACT(MEM)),
       STRUC(MBNTY(MEM)).NE.0,A1(MEM)="DELAY",A2(MEM)=ANMS(SUP(MBNTY(MEM))),
       A3(MEM)=ANMS(@CLASATR(STRUC(MBNTY(MEM)))(MBNTY(MEM))),RSTRUC,
                                  MBNTY(MEM)=STRUC(MBNTY(MEM)))

SRCHREC(CODE.EQ.3,SUP.EQ.'WAIT',IDNAME.EQ.IDNAME(CURSEG(MEM)))  -->
                              CHKACTN(CURSEG(MEM)=REC(MEM))

SRCHFAIL(CODE.EQ.3) --> NEWACTN(DURATION(CURSEG(MEM))=TEMPSEG(MEM),
       LASTREC('CNDL')=LASTREC('CNDL')+1,MBNTY(CURSEG(MEM))=MBNTY(MEM),
                                  @LASTREC('CNDL')('CNDL')=CURSEG(MEM))

CHKACTN(SUP(LASTACT(MEM)).NE.'TEMPACT') --> NEXTACT(SUCC(LASTACT(MEM))=
                                       CURSEG(MEM),-CHKPOINT(MEM))

CHKACTN(SUP(LASTACT(MEM)).EQ.'TEMPACT') --> DELACT

NEWACTN(SUP(LASTACT(MEM)).NE.'TEMPACT') --> NEXTACT(IDNO(CURSEG(MEM))=
       IDACT(MEM),IDACT(MEM)=IDACT(MEM)+1,SUCC(LASTACT(MEM))=CURSEG(MEM),
       LASTREC('ACTL')=LASTREC('ACTL')+1,@LASTREC('ACTL')('ACTL')=
       CURSEG(MEM),-CURSEG(MEM))

NEWACTN(SUP(LASTACT(MEM)).EQ.'TEMPACT') --> STARTACT(CSG=CURSEG(MEM),
       LAM=LASTACT(MEM),@ALPOS(LAM)('ACTL')=CSG,@XY(LAM)(POINT(LAM))=CSG,
              LASTACT(MEM)=CSG,-CURSEG(MEM))
```

```
DELACT --> DELACT1(@XY(LASTACT(MEM))(POINT(LASTACT(MEM)))=CURSEG(MEM),
       COUNT=ALPOS(LASTACT(MEM)),NEXT=COUNT+1,LAST=COUNT-1,
                                    LASTACT(MEM)=@LAST(SEG)('ACTL'))

DELACT1(COUNT.LT.LASTREC('ACTL')) --> DELACT1(@COUNT(SEG)('ACTL')=
                          @NEXT(SEG)('ACTL'),COUNT=NEXT,NEXT=NEXT+1)

DELACT1(COUNT.EQ.LASTREC('ACTL')) --> NEXTACT(LASTREC('ACTL')=
                                       LASTREC('ACTL')-1,-CURSEG(MEM))

CONTNUSG(CHKPOINT(MEM).EQ.6) --> REPLY1(MBNTY(MEM)=STRUC(MBNTY(MEM)),
                                  -CHKPOINT(MEM),X3(MEM)=STRUC(MBNTY(MEM)))

CKDONE(CHKPOINT(MEM).EQ.6) --> REPLY1(-CHKPOINT(MEM),RDONE)

LEAVS /PUNCD(CHKPOINT(MEM).EQ.7) --> NACT1('LEAV')

WORD(CHKPOINT(MEM).EQ.7,¬LEAVSW*,$'ACTION'|$'UNKNOWN') -->
                                                    NACT1(SUP(WORD))

./ NACT1  -->  NACT2(%NACT1)

WORD($'MOBENTY') NACT1  -->  NACT2(%NACT1)

NACT2 ADJ('THE')  -->  NACT2

NACT2 WORD($'OBJENTY'|$'UNKNOWN') -->  NACT2A(%NACT2,GOAL=SUP(WORD),
                                                 SUP(GOAL)='OBJENTY')

NACT2 /WORD(¬$'OBJENTY',¬$'UNKNOWN') --> NACT2A(%NACT2)

NACT2A PREP($'LOCDESCR') -->  TMPS(SUP(WORD),TEMPSEG(MEM)=SEG),
                               NACT3(%NACT2A,LOCATION=TEMPSEG(MEM))

NACT3 ADJ('THE')  -->  NACT3

NACT3($'UNKNOWN') -->  NACT3(SUP(SUP)='EVENT',A1(MEM)=ANMS(SUP),RACTION,
                                  SUP(SUP)='ACTIVITY')

NACT3($'ACTION') WORD(¬$'STATENTY') --> REPLY(SUP(WORD),ERR(MEM)=13,RERR)

NACT3($'ACTION') WORD($'STATENTY') -->  SEARCH(LIST='STAL',CODE=5,
            SUP(MEM)=SUP(WORD),CURSEG(MEM)=NACT3,MBNTY(NACT3)=MBNTY(MEM))

SRCHREC(CODE.EQ.5,SUP.EQ.SUP(MEM)) -->
   SEARCH(LOCOBJ(LOCATION(CURSEG(MEM)))=REC(MEM),LIST='ACTL',CODE=6)

SRCHREC(CODE.EQ.6,SUP=SUP(CURSEG(MEM)),MBNTY.EQ.MBNTY(CURSEG(MEM)),
```

```
  LOCATION.EQ.LOCATION(CURSEG(MEM))) --> CHKACTN(CURSEG(MEM)=REC(MEM),RDONE)

SRCHFAIL(CODE.EQ.6,CURSEG(MEM)$'ACTIVITY') -->
                                        REPLY1(GETDIST(MEM)='DURATION')

SRCHFAIL(CODE.EQ.6,CURSEG(MEM)$'EVENT') --> NEWACTN(RDONE)

CKDONE(CHKPOINT(MEM).EQ.7,CURSEG(MEM)$'ACTIVITY') -->
                                                DISTSEG3(SUP(REC(MEM)))

CONTNUSG(CHKPOINT(MEM).EQ.7,CURSEG(MEM)$'ACTIVITY') --> TMPS('PARAMNO',
                                    NUM(PNUM(ASNDISTR(MEM))),TEMPSEG(MEM)=SEG),
   TABL1('TYPTABL',FNID=MFNID(MEM),MFNID(MEM)=MFNID(MEM)+1,XYLAST=100,
       FNARG=TEMPSEG(MEM),DORC="D",TBL1(MEM)=SEG,MEAN(REC(MEM))=SEG,
       DURATION(CURSEG(MEM))=REC(MEM),SUP(REC(MEM)).NE.'EXPON',
                                    SUP(REC(MEM)).NE.'CONSTANT'),
   TABL2('TYPTABL',FNID=MFNID(MEM),MFNID(MEM)=MFNID(MEM)=1,TBL2(MEM)=SEG,
       FNARG=TEMPSEG(MEM),DORC="D",XYLAST=100,RANGE(REC(MEM))=SEG,
       SUP(REC(MEM)).NE.'UNIFORM',STDEV(REC(MEM))=SEG,-RANGE(REC(MEM)))

TABL1(CHKPOINT(MEM).EQ.7) --> SEARCH(LIST='MOBL',CODE=7,X5(MEM).EQ.2,
       SUP(REC(MEM))=TIMEUNIT(MEM),NUM(REC(MEM))=MEAN(REC(MEM)),
                                                -MEAN(REC(MEM)))

SRCHFAIL(CODE.EQ.7)  -->  NEWACTN(-TBL1(MEM),-TBL2(MEM),RDONE)

SRCHREC(CODE.EQ.7,STRUC.EQ.MBNTY(MEM),SUP.EQ.SUP(MBNTY(MEM))) -->
   REPLY1(A2(MEM)=ANMS(@CLASATR(MBNTY(MEM))(REC(MEM))))

NMBER(CHKPOINT(MEM).EQ.8) --> NUMB99(NUM(NMBER),CHKPOINT(MEM)=99)

NUMB99 WORD  -->  NUMB99

NUMB99 PUNCD  -->  NUMB99

NUMB99(RCHKJ)  -->  NUMB2(SUP(TIMEUNIT(MEM),CHKPOINT(MEM)=8),NUM(NUMB99))

NUMB99 NMBER  --> NUMB2(SUP(TIMEUNIT(MEM)),NUM(NUMB99),NUM2=NUM(NMBER),
                                        CHKPOINT(MEM)=8)

NUMB2 -->  SETBL1(XY=XYLAST(TBL1(MEM))+1,@XY(SEG)(TBL1(MEM))=REC(MEM),
       XY=XY+1,@XY(SEG)(TBL1(MEM))=NUMB2,XYLAST(TBL1(MEM))=XY,
                            XY(MEM)=XY,TBL2(MEM).NE.0),
   TMPS(SUP(NUMB2),NUM=NUM2(NUMB2),-NUM2(NUMB2),TEMPSEG(MEM)=SEG),
   SETBL2(XY=XYLAST(TBL2(MEM))+1,@XY(SEG)(TBL2(MEM))=REC(MEM),
       XY=XY+1,@XY(SEG)(TBL2(MEM))=TEMPSEG(MEM),XYLAST(TBL2(MEM))=XY)

SETABL1(CHKPOINT(MEM).EQ.8) -->  SEARCH(LIST='MOBL',PREVREC=REC(MEM),
```

```
        CODE=7,X5(MEM).EQ.2,@XY(MEM)(TBL1(MEM))=NUM(@XY(MEM)(TBL1(MEM))))

SRCHREC(CODE.EQ.8,SUP.NE.SUP(CURMOB(MEM))) --> TMPS('AT',TEMPSEG(MEM)=SEG,
                                                LOCOBJ=@11('STAL')),
   ARRIVS(LOCATION=TEMPSEG(MEM),CURMOB(MEM)=REC(MEM))

SRCHFAIL(CODE.EQ.8) -->  PROBTIM(SUP=TIMEUNIT(MEM),RPRBTIM,NUM=X3(MEM),
                                                PROBTIME(MEM)=SEG)

PROBTIM -->  SEARCH(COUNT(MEM)=11,LIST='SCRL',CODE=9)

SRCHREC(CODE.EQ.9,REC(MEM).EQ.SUCC(@COUNT(MEM)('ACTL'))) --> NXTSUCC

SRCHFAIL(CODE.EQ.9) -->  NXTSUCC(LASTREC('SCRL')=LASTREC('SCRL')+1,
                         @LASTREC('SCRL')('SCRL')=SUCC(@COUNT(MEM)('ACTL')))

NXTSUCC(COUNT(MEM).LT.LASTREC('ACTL')) -->  SEARCH(LIST='SCRL',CODE=9,
       XY=COUNT(MEM),COUNT(MEM)=COUNT(MEM)+1,SUP(@XY(SEG)('ACTL')).NE.
                                  'ARRIV',PRED(@XY(SEG)('ACTL'))=1)

NXTSUCC(COUNT(MEM).GE.LASTREC('ACTL')) -->  REPLY1(RDONE,RENDPGM,
                                A1(MEM)=ANMS(@11('STAL')),-CURSEG(MEM))

SRCHREC      -->       SEARCH2
```

APPENDIX D

THE CONTROL PROGRAM - QUEST

```
      SUBROUTINE QUEST(*)
C
C *****  INITIALIZATION  *****
C
      IMPLICIT INTEGER*2 (A-Z)
      REAL*8 WRD(10),DBLANK/' '/,DWD/'DECIMAL'/,
     X PWD/'PERCENT'/,QWD/'QUANTITY'/,UWD/'UNIT'/,
     X CWD/'CONSUMP'/,BWD/'CAPACITY'/,DVAL
      INTEGER*2 AM(8)/31,32,33,34,35,36,37,38/
      DATA ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE,TEN,ELEVEN,
     X TWELVE/1,2,3,4,5,6,7,8,9,10,11,12/,
     X STONUM/40/,NUMNAM/41/,CNDTYP/43/,CNDNAM/44/,SEQNUM/45/
      COMMON /MAIN/ COLZ,COL(80),TRMPNZ,TRMPNT(255),NAMEZ,NAMEX(255)
      COMMON /MAIN/ STYPE,RULE,SEGMNT,CLIST,IDNOSG,MEMORY,SSGMNT
      COMMON /MAIN/ MAXJ,J,TRLEVR,TRLEVS,RULENO
      COMMON /MAIN/ COLON,SLASH,MINUS,COMMA,PERIOD,BLANK,TTT,LBSIGN
      COMMON /MAIN/ ZE,ZBLANK,DUMMY
      INTEGER*4 OUT6,RTERM,WTERM,FLNUMB
      COMMON /MAIN/ OUT6,RTERM,WTERM,FLNUMB
      LOGICAL TERMSW,PRTSW,KEEPCL,TRNS2,TR1,TR2,TR3,TR4,TR5,TR6
      COMMON /MAIN/ TERMSW,PRTSW,KEEPCL,TRNS2,TR1,TR2,TR3,TR4,TR5,TR6
      LOGICAL OK,DONE,FSTCND,THEEND,TR7
      REAL*8 AWORD,BWORD,CWORD,DWORD
      COMMON /QUESTN/ AWORD,BWORD,CWORD,DWORD
      COMMON /QUESTN/ OK,DONE,FSTCND,THEEND
      COMMON /QUESTN/ ANUM,AJNUM,AERR,CHKPNT
      COMMON /QUESTN/ AME1,AME2,AME3,AME4,AME5,AME6,YES,NO
      COMMON /QUESTN/ XME1,XME2,XME3,XME4,XME5,XME6
      NAMELIST /P/ TR1,TR2,TR3,TR4,TR5,TR6,TR7,OUT6,TRLEVR,TRLEVS
C
C *****  QUESTIONAIRE STARTS HERE  *****
C
      CAPCTY = LOCKUP(BWD,AKTE)
      CNSUMP = LOOKUP(CWD,AKTE)
      UNITV = LOOKUP(UWD,AKTE)
      PERCTV = LOOKUP(PWD,AKTE)
      DECMV = LOOKUP(DWD,AKTE)
      QUANV = LOOKUP(QWD,AKTE)
      TR7 = .FALSE.
```

```
      FSTCND = .TRUE.
      WRITE(WTERM,15)
   15 FORMAT(' ENTER OPTIONAL DATA FOR QUESTIONAIRE SESSION')
      READ(RTERM,P)
      IF(.NOT.TR7) GO TO 20
      WRITE(WTERM,17)
   17 FORMAT(' REMARK: MAXIMUM OF 8 CHARACTERS FOR NAMES, MUST BE UNIQUE
     X FOR NEW NAMES')
   20 WRITE(WTERM,21)
   21 FORMAT(' ENTER NAME FOR SYSTEM TO BE SIMULATED')
      CALL DECODE(ONE,&305)
      IF(OK) GO TO 25
      IF(HVAL(AERR,MEMORY).NE.0) GO TO 20
      CALL ERROR(J,3)
      GO TO 20
C
C   **** ENTER STATIONARY ENTITIES  *****
C
   25 WRITE(WTERM,26)
   26 FORMAT(' ENTER NAME OF FIRST STATIONARY ENTITY')
      GO TO 35
   30 WRITE(WTERM,31)
   31 FORMAT(' RE-ENTER NAME OF STATIONARY ENTITY')
   35 CALL DECODE(ONE,&305)
      IF(DONE) GO TO 50
      IF(OK) GO TO 32
      IF(HVAL(AERR,MEMORY).NE.0) GO TO 30
      CALL ERROR(J,3)
      GO TO 30
   32 BWORD = DVAL(AME1,MEMORY)
   33 WRITE(WTERM,34) BWORD
   34 FORMAT(' HOW MANY IDENTICAL ',A8,' ENTITIES ARE THERE ?')
      CALL PSTORE(QUANV,STONUM,MEMORY)
      CALL DECODE(ONE,&305)
      IF(OK) GO TO 36
      CALL ERROR(J,3)
      GO TO 33
   36 WRITE(WTERM,37) BWORD
   37 FORMAT(' HOW MANY "CUSTOMERS" (MAX) CAN EACH ',A8,' PROCESS SIMULT
     XANEOUSLY ?')
      CALL PSTORE(CAPCTY,STONUM,MEMORY)
      CALL PSTORE(UNITV,NUMNAM,MEMORY)
      CALL DECODE(ONE,&305)
      IF(OK) GO TO 38
      CALL ERROR(J,3)
      GO TO 36
C
C *****  GET OPTIONAL ATTRIBUTES FOR STATIONARY ENTITIES  *****
```

```
C
   38 CALL GETOPT(WTERM,BWORD,&305)
      WRITE(WTERM,40) BWORD
   40 FORMAT(' ',A8,' COMPLETE...NAME NEXT STATIONARY ENTITY ...OR TYPE
     X"NONE" IF DONE')
      CALL HSTORE(ONE,CHKPNT,MEMORY)
      GO TO 35
C
C  *****  ENTER MOBILE ENTITIES  *****
C
   52 WRITE(WTERM,53)
   53 FORMAT(' ENTER NAME OF FIRST MOBILE ENTITY')
      GO TO 56
   54 WRITE(WTERM,55)
   55 FORMAT(' RE-ENTER NAME OF MOBILE ENTITY')
   56 CALL DECODE(ONE,&305)
      IF(DONE) GO TO 100
      IF(OK) GO TO 58
      IF(HVAL(AERR,MEMORY).NE.0) GO TO 54
      CALL ERROR(J,3)
      GO TO 54
   58 BWORD = DVAL(AME2,MEMORY)
   59 WRITE(WTERM,60) BWORD
   60 FORMAT(' HOW MANY SERVICE UNITS DOES EACH ',A8,' REQUIRE ?')
      CALL PSTORE(CSUMP,STONUM,MEMORY)
      CALL PSTORE(UNITV,NUMNAM,MEMORY)
      CALL DECODE(ONE,&305)
      IF(OK) GO TO 63
      CALL ERROR(J,3)
      GO TO 59
   63 WRITE(WTERM,64) BWORD
   64 FORMAT(' IS THERE A SUBCLASSIFICATION OF ',A8,',MAX OF 1 ?')
      CALL DECODE(ONE,&305)
      IF(DONE) GO TO 80
      IF(OK) GO TO 68
      CALL ERROR(J,3)
      GO TO 63
   68 WRITE(WTERM,69) BWORD
   69 FORMAT(' NAME THE ATTRIBUTE THAT DISTINGUISHES EACH ',A8)
      CALL DECODE(ONE,&305)
      IF(OK) GO TO 80
      IF(HVAL(AERR,MEMORY).NE.0) GO TO 68
      CALL ERROR(J,3)
      GO TO 68
   80 WRITE(WTERM,81) BWORD
   81 FORMAT(' OPTIONAL ATTRIBUTES FOR ALL ',A8)
      GO TO 84
   82 AWORD = DVAL(AME1,MEMORY)
```

```
      WRITE(WTERM,83) BWORD,AWORD
   83 FORMAT(' OPTIONAL ATTRIBUTES FOR ALL ',A8,' REGARDLESS OF ',A8)
C
C *****  GET OPTIONAL ATTRIBUTES FOR MOBILE ENTITIES  *****
C
   84 CALL GETOPT(WTERM,BWORD,&305)
      WRITE(WTERM,90) BWORD
   90 FORMAT(' ',A8,' COMPLETED..NEXT MOBILE ENTITY NAME..OR "NONE"')
      CALL HSTORE(TWO,CHKPNT,MEMORY)
      GO TO 56
C
C  ***** ACTION BUILDER *****
C
  100 THEEND = .FALSE.
      FSTCND = .TRUE.
      IF(.NOT.TR7) GO TO 110
  110 BWORD = DVAL(AME1,MEMORY)
      AWORD = DVAL(AME2,MEMORY)
  111 WRITE(WTERM,112) AWORD,BWORD,AWORD
  112 FORMAT(' CONSTRUCT ACTION SEQUENCE FOR ',A8/' PROBABILITY DISTRIBU
     XTIONS AVAILABLE: EXPONENTIAL, UNIFORM, NORMAL. CONSTANT(NO DEVIATI
     XON)'/'  ENTER TYPE OF DISTRIBUTION FOR TIME BETWEEN ARRIVALS OF ',
     X A8,' AT THE ',A8)
      CALL DECODE(ONE,&305)
      IF(OK) GO TO 113
      CALL ERROR(J,3)
      GO TO 111
  113 WRITE(WTERM,114)
  114 FORMAT(' GIVE THE TIME UNIT TO BE USED (HOUR,SECOND,ETC)')
      CALL DECODE(ONE,&305)
      IF(OK) GO TO 115
      IF(HVAL(AERR,MEMORY).NE.0) GO TO 113
      CALL ERROR(J,3)
      GO TO 113
C
C  *****    TOP OF THE ACTION LOOP *****
C
  115 IF(THEEND) GO TO 300
      DO 116 I=1,6
      AME = AM(I)
  116 WRD(I) = DVAL(AME,MEMORY)
      PT     = HVAL(XME1,MEMORY)
      SUBTYP = HVAL(XME2,MEMORY)
      SUPTYP = HVAL(XME3,MEMORY)
      ITYP   = HVAL(XME4,MEMORY)
      WRITE(WTERM,118)
  118 FORMAT(' FOLLOWING THE ACTION:')
      WRITE(WTERM,119) (WRD(I),I=1,6)
```

```
  119 FORMAT(' ',6A8)
      IF(ITYP.NE.2) GO TO 125
      PCT = HVAL(XME5,MEMORY)
      SEQ = HVAL(XME6,MEMORY)
      WRITE(WTERM,123) PCT,SEQ
  123 FORMAT(' VIA THE ',I2,'% LEG, (ROUTE',I2,')')
  125 IF(SUBTYP.EQ.0) GO TO 132
      WRD(9) = DVAL(ANMS,SUBTYP)
  128 WRITE(WTERM,129) WRD(9),WRD(PT)
  129 FORMAT(' DOES ROUTING DEPEND ON ',A8,' OF ',A8,'?')
      CALL HSTORE(FOUR,CHKPNT,MEMORY)
      CALL DECODE(ONE,&305)
      IF(DONE) GO TO 132
      IF(OK) GO TO 115
      CALL ERROR(J,3)
      GO TO 128
  132 IF(.NOT.FSTCND) GO TO 136
      FSTCND = .FALSE.
C
C *****  SPECIFY CONDITIONALS  *****
C
      WRITE(WTERM,133)
  133 FORMAT(' FORMAT FOR CONDITIONAL SPECIFICATION:'/
     X'    1. <TYPE OF CONDITIONAL>,<IDENT NAME> FOR NEW CONDITIONAL'/
     X'    2. <IDENT NAME> ONLY FOR ROUTE TO KNOWN CONDITIONAL'/
     X'    3. "NONE" IF NONE ')
  136 WRITE(WTERM,137) WRD(PT)
  137 FORMAT(' SPECIFY CONDITIONAL (OR "NONE") FOR ',A8)
      CALL HSTORE(FIVE,CHKPNT,MEMORY)
      CALL DECODE(ONE,&305)
      IF(DONE) GO TO 155
      IF(OK) GO TO 140
      IF(HVAL(AERR,MEMORY).NE.0) GO TO 136
      CALL ERROR(J,3)
      GO TO 136
  140 CWORD = DVAL(AME1,MEMORY)
      GO TO (115,115,143,148,115),ITYPE
  143 WRITE(WTERM,144) CWOPD
  144 FORMAT(' ENTER PERCENT OF EACH BRANCH FROM ',A8/
     X'     EXAMPLE:  30,30,40')
      CALL DECODE(ONE,&305)
      IF(OK) GO TO 115
      IF(HVAL(AERR,MEMORY).NE.0) GO TO 143
      CALL ERROR(J,3)
      GO TO 143
  148 WRITE(WTERM,149) CWORD
  149 FORMAT(' ENTER: LOCATION OF WAIT, ENTITY CAUSING DELAY, AND CONDIT
     XION REQUIRED TO CONTINUE')
```

```
      CALL DECODE(ONE,&305)
      IF(OK) GO TO 115
      IF(HVAL(AERR,MEMORY).NE.0) GO TO 148
      CALL ERROR(J,3)
      GO TO 148
  155 SUPTYP = HVAL(XME3,MEMORY)
      IF(SUPTYP.EQ.0) GO TO 170
      WRD(9) = DVAL(ANMS,SUPTYP)
  162 WRITE(WTERM,163) WRD(PT),WRD(9),WRD(PT)
  163 FORMAT(' WILL NEXT ACTION BE FOR ALL ',A8,' VICE ',2A8,' ONLY ?')
      CALL HSTORE(SIX,CHKPNT,MEMORY)
      CALL DECODE(ONE,&305)
      IF(DONE) GO TO 170
      IF(OK) GO TO 155
      CALL ERROR(J,3)
      GO TO 162
  170 II = PT + 1
      DO 172 I = II,4
  172 WRD(I) = DBLANK
  173 WRITE(WTERM,174) (WRD(I),I=1,4)
  174 FORMAT(' NAME NEXT ACTION AND LOCATION FOR ',4A8)
      CALL HSTORE(SEVEN,CHKPNT,MEMORY)
      CALL DECODE(ONE,&305)
      IF(DONE) GO TO 115
      IF(OK) GO TO 176
      IF(HVAL(AERR,MEMORY).NE.0) GO TO 173
      CALL ERROR(J,3)
      GO TO 173
  176 WRD(8) = DVAL(AME1,MEMORY)
  178 WRITE(WTERM,179) WRD(8)
  179 FORMAT(' ENTER DISTRIBUTION TYPE FOR DURATION OF ',A8)
      CALL DECODE(ONE,&305)
      IF(DONE) GO TO 115
      IF(OK) GO TO 182
      CALL ERROR(J,14)
      GO TO 178
  182 WRD(9) = DVAL(AME2,MEMORY)
      WRITE(WTERM,183) WRD(8),WRD(9),WRD(PT)
  183 FORMAT(' DOES DURATION OF ',A8,' DEPEND ON ',A8,' OF ',A8,'?')
      CALL DECODE(ONE,&305)
      IF(DONE) GO TO 115
      IF(OK) GO TO 186
      CALL ERROR(J,3)
      GO TO 182
  186 CALL HSTORE(EIGHT,CHKPNT,MEMORY)
      I = HVAL(XME5,MEMORY)
  188 WRD(9) = DVAL(AME2,MEMORY)
      GO TO (210,220,230,240),I
```

```
  210 WRITE(WTERM,212) WRD(8),WRD(9),WRD(PT)
  212 FORMAT(' GIVE MEAN OF EXPONENTIAL DIST. FOR DURATION OF ',3A8)
      GO TO 250
  220 WRITE(WTERM,222) WRD(8),WRD(9),WRD(PT)
  222 FORMAT(' ENTER CONSTANT ',A8,' TIME FOR ',2A8)
      GO TO 250
  230 WRITE(WTERM,232) WRD(8),WRD(9),WRD(PT)
  232 FORMAT(' FOR NORMAL DISTRIBUTION OF ',3A8,' GIVE: MEAN, STANDARD D
     XEVIATION')
      GO TO 250
  240 WRITE(WTERM,242) WRD(8),WRD(9),WRD(PT)
  242 FORMAT(' FOR UNIFORM DIST. OF ',3A8,' GIVE MEAN, ABS(RANGE/2)')
  250 CALL DECODE(ONE,&305)
      IF(DONE) GO TO 115
      IF(OK) GO TO 188
      CALL ERROR(J,3)
      GO TO 188
C
CC   *****  THE END  *****
C
  300 DWORD = DVAL(AME1,MEMORY)
      WRITE(WTERM,301) DWORD
  301 FORMAT(' DATA STRUCTURE FOR ',A8,' SIMULATION IS COMPLETE')
      RETURN
  305 WRITE(WTERM,306)
  306 FORMAT(' ABNORMAL END OF QUESTIONAIRE')
      RETURN 1
      END

C**********************************************************************
C
C *****  OPTIONAL ATTRIBUTE ROUTINE  *****


      SUBROUTINE GETOPT(WTERM,SWORD,*)
      INTEGER*2 APTE/1/,ANMS/10/,ONE/1/
      INTEGER*4 WTERM
      REAL*8 AWORD,BWORD,CWORD,DWORD,SWORD
      LOGICAL CK,DONE,FSTCND,THEEND
      COMMON /QUESTN/ AWORD,BWORD,CWORD,DWORD
      COMMON /QUESTN/ OK,DONE,FSTCND,THEEND
      IF(FSTCND) WRITE(WTERM,2)
    2 FORMAT(' IDENTIFY AND ASSIGN VALUES OR NAMES TO OPTIONAL ATTRIBUTE
     XS; EXAMPLES:'/'   LOCATION IS THE HARBOR'/'   WEIGHT = 1000 POUNDS
     X'/'   SPEED IS FAST')
      FSTCND = .FALSE.
    5 WRITE(WTERM,10) SWORD
   10 FORMAT(' ENTER OPTIONAL ATTRIBUTE FOR ',A8,' (OR "NONE")')
```

```
      CALL DECODE(ONE,&30)
      IF(DONE) RETURN
      IF(.NOT.OK) CALL ERROR(J,15)
      GO TO 5
   30 RETURN 1
      END

C*****************************************************
C
C *****  ERROR ROUTINE  *****
C     ALL ERROR MESSAGES ARE FOUND IN HERE

      SUBROUTINE ERROR(INDX,TYP)
      IMPLICIT INTEGER*2 (A-Z)
      COMMON /MAIN/ COLZ,COL(80),TRMPNZ,TRMPNT(255),NAMEZ,NAMEX(255)
      COMMON /MAIN/ STYPE,RULE,SEGMNT,CLIST,IDNOSG,MEMORY,SSGMNT
      COMMON /MAIN/ MAXJ,J,TRLEVR,TRLEVS,RULENO
      COMMON /MAIN/ COLON,SLASH,MINUS,COMMA,PERIOD,BLANK,TTT,LBSIGN
      COMMON /MAIN/ ZF,ZBLANK,DUMMY
      INTEGER*4 OUT6,RTERM,WTERM,FLNUMB,TYP
      COMMON /MAIN/ OUT6,RTERM,WTERM,FLNUMB
      DIMENSION ERRIND(80), ERR(20,20)
      DATA MARK/'$'/,ERR/
     1'ATTRIBUTE IS NOT IN CLASS INDICATED      ',
     2'COMPLETE REPLY DETECTED AT $             ',
     3'INCOMPLETE REPLY AT $...RE-ENTER         ',
     4'INVALID LOCATION ATTRIBUTE               ',
     5'RECODE NOT FOUND ON CORRECT LIST.        ',
     6'INVALID ATTRIBUTE TYPE OR NAME. RE-ENTER',
     7'INCORRECT FUNCTION DEFINITION.           ',
     8'NAME NOT FOUND                           ',
     9'UNDEFINED PRINT COMMAND                  ',
    10'RECORD NUMBER NEGATIVE OR > MAXLN        ',
    11'VALUES ENTERED DO NOT TOTAL 100%,REENTER',
    12'NOT A VALID ACTION DEFINITION            ',
    13'INVALID STATIONARY ENTITY NAME           ',
    14'INVALID DISTRIBUTION NAME                ',
    15'COULD NOT DECODE OPTIONAL ATTRIBUTE      ',
    16'INVALID MOBILE ENTITY NAME               ',
    17'INVALID TIME UNIT NAME...RE-ENTER        ',
    18'RULE NOT FOUND                           ',
    19'                                         ',
    20'                                        '/
C                                          ERROR
      WRITE(WTERM,101) (COL(I),I=1,80)
  101 FORMAT(/' ',80A1)
      DO 105 I=1,80
```

```
  105 ERRIND(I) = BLANK
      ERRIND(INDX) = MARK
      ITYP = TYP
      WRITE(WTERM,102) (ERRIND(I),I=1,80)
  102 FORMAT(' ',80A1)
      WRITE(WTERM,110) (ERR(I,ITYP),I=1,20)
  110 FORMAT(' **',20A2)
      RETURN
      END

C*************************************************************
C
C   *****  PRTCLS PRINTS THE SUPERSET (SUP) "CHAIN"  *****

      SUBROUTINE PRTCLS(SEG,ATTR,WTERM)
      INTEGER*2 SEG,PSEG,ATTR,ANMS/10/
      INTEGER*4 WTERM
      REAL*8 RECNAM,SUPNAM
      PSEG = HVAL(ATTR,SEG)
      IF(PSEG.EQ.0) RETURN
      WRITE(WTERM,1)
    1 FORMAT(' SUPERSET CHAIN:')
      SUPNAM = DVAL(ANMS,PSEG)
    2 PSEG = HVAL(ATTR,PSEG)
      IF(PSEG.EQ.0) RETURN
      RECNAM = SUPNAM
      SUPNAM = DVAL(ANMS,PSEG)
    5 WRITE(WTERM,10) RECNAM,SUPNAM
   10 FORMAT('    ',A8,'->',A8)
      GO TO 2
      END
```

APPENDIX E

RULE-CALLABLE FORTRAN ROUTINES

```
C                                  *** ROUTINES ***
C
C    THE ROUTINES EXIST IN SUBROUTINE CRSEG, AND ARE INITIATED
C    FROM THE RULES BY BEING NAMED INSIDE PARENTHESES.
C
  480 I = ADC + 1
      IF (.NOT.TR6)  GO TO 484
      WRITE (WTERM,482)  ADC
  482 FORMAT(' ROUTINE',I3)
  484 GO TO (1200,1200,1200,1200,1200,1200,1200,1200,1200,1200,1200,
     X5010,5020,5030,5040,5050,5060,5070,5080,5090,5100,5110,5120,5130,
     X5140,5150,5160,5170,5180,5190), I
C
C                                          ROUTINE 11  RERR
 5010 ERRNUM = HVAL(AERR,MEMORY)
      CALL ERROR(J,ERRNUM)
      CALL PRTCLS(SEGMNT,APTE,WTERM)
      GO TO 1200
C
C                                          ROUTINE 12  RSUP
 5020 DWORD = DVAL(ANMS,HVAL(EIGHT,SEGMNT))
      SUPVAL = HVAL(APTE,SEGMNT)
      CALL SETSUP(DWORD,DWORD,SUPVAL,SEGMNT,J)
      IF(DWORD.NE.0.) GO TO 1200
      CALL PRTCLS(SEGMNT,APTE,WTERM)
      J = MAXJ
      GO TO 1300
C
C                                          ROUTINE 13  RCHKJ
 5030 IF(J.GE.(MAXJ-1)) GO TO 1200
      GO TO 1300
C
C                                          ROUTINE 14  RSETSUP
 5040 N = J-1
      J = HVAL(AJNUM,SEGMNT)
      CALL COLECT(CWORD)
      DWORD = CWORD
      IF(COL(N).NE.SSS) GO TO 5041
      COL(N) = BLANK
      J = HVAL(AJNUM,SEGMNT)
      CALL COLECT(DWORD)
 5041 CALL SETSUP(CWORD,DWORD,ZERO,SEGMNT,J)
```

```
      J = N + 1
      GO TO 1200
C
C                                        ROUTINE 15  RDONE
 5050 DONE = .TRUE.
      GO TO 1200
C
C                                        ROUTINE 16  RGETJ
 5060 CALL HSTORE(J,AJNUM,SEGMNT)
      GO TO 1200
C
C                                        ROUTINE 17  RCKEND
 5070 IF(J.LT.(MAXJ-1)) GO TO 1200
      GO TO 1300
C
C                                        ROUTINE 18  RPRMT1
 5080 DWORD = DVAL(AME1,MEMORY)
      CWORD = DVAL(AME2,MEMORY)
      WRITE(WTERM,5081) DWORD,CWORD
 5081 FORMAT(' NOW ENTER ',A8,' DATA UNIQUE TO EACH ',A8)
      I = 0
      GO TO 1200
C
C                                        ROUTINE 19  RPRMT2
 5090 I = I + 1
      WRITE(WTERM,5091) DWORD,CWORD,I
 5091 FORMAT(' ENTER NAME OR VALUE OF ',A8,' FOR ',A8,' NUMBER',I3)
      READ(RTERM,5092) BWORD
 5092 FORMAT(A8)
      MVAL = LOOKUP(BWORD,AKTE)
      IF(MVAL.EQ.0) MVAL = ENTER(BWORD,AKTE)
      CALL PSTORE(MVAL,XME2,MEMORY)
      WRITE(WTERM,5093) CWORD,DWORD,BWORD
 5093 FORMAT(' WHAT PERCENTAGE OF ',A8,' HAVE ',A8,' = ',A8,'?')
 5094 MAXJ = 0
      CALL NXTCHR(&1300)
      CALL CONVRT(NUM)
      CALL HSTORE(NUM,XME3,MEMORY)
      GO TO 1200
C
C                                        ROUTINE 20  RPRMT3
 5100 WRITE(WTERM,5101)
 5101 FORMAT(' GIVE STANDARD DEVIATION (NORMAL), OR PLUS/MINUS RANGE VAL
     XUE (UNIFORM)')
      GO TO 5094
C
C                                        ROUTINE 21  ROK
 5110 OK = .TRUE.
```

```
      GO TO 1200
C
C                                        ROUTINE 22  RSTRUC
 5120 BWORD = DVAL(AME1,MEMORY)
      CWORD = DVAL(AME2,MEMORY)
      DWORD = DVAL(AME3,MEMORY)
      WRITE(WTERM,5122) CWORD,DWORD,CWORD,BWORD
 5122 FORMAT(' MIGHT ',A8,' OTHER THAN ',2A8,' PASS THROUGH ',A8,'?')
 5123 READ(RTERM,5124) REPY
 5124 FORMAT(A1)
      IF(REPY.EQ.YES) GO TO 1200
      GO TO 1300
C
C                                        ROUTINE 23  RMEAN
 5130 DWORD = DVAL(AME1,MEMORY)
      WRITE(WTERM,5131) DWORD
 5131 FORMAT(' ENTER AVERAGE VALUE(MEAN) OF ',A8,' DISTRIBUTION')
      GO TO 5094
C
C                                        ROUTINE 24  RACTION
 5140 DWORD = DVAL(AME1,MEMORY)
      WRITE(WTERM,5141) DWORD
 5141 FORMAT(' DOES ACTION ',A8,' HAVE A DURATION OR DELAY TIME ?')
      GO TO 5123
C
C                                        ROUTINE 25  RPRBTIM
 5150 DWORD = DVAL(ANMS,HVAL(ONE,SEGMNT))
      WRITE(WTERM,5151) DWORD
 5151 FORMAT(' GIVE TOTAL PROBLEM TIME IN ',A8,' UNITS')
      GO TO 5094
C
C                                        ROUTINE 26  RENDPGM
 5160 THEEND = .TRUE.
      GO TO 1200
C
C                                        ROUTINE 27  RSTOP
 5170 WRITE(WTERM,5172).
 5172 FORMAT(' OK ?')
      READ(RTERM,5124)  REPY
      IF (REPY.EQ.YES)  OK = .TRUE.
      WRITE(WTERM,5174)
 5174 FORMAT(' DONE ?')
      READ(RTERM,5124)  REPY
      IF (REPY.EQ.YES)  DONE = .TRUE.
      WRITE(WTERM,5176)
 5176 FORMAT(' KEEP CURSEG ?')
      READ(RTERM,5124)  REPY
      IF (REPY.NE.YES)  GO TO 1300
```

```
      GO TO 1200
         ********* END OF ROUTINES (IN CRSEG) *********
```

LIST OF REFERENCES

1. RAND Corporation Memorandum RM-5129-PR, Programming by Questionaire: How to Construct a Program Generator, by P.M. Oldfather, A.S. Ginsberg, and H.P. Markowitz, November 1966.

2. RAND Corporation Memorandum RM-4460-PR, Programming by Questionaire, by P.M. Oldfather, A.S. Ginsberg, and H.P. Markowitz, April 1965.

3. Simmons, R.F., Natural Language Question Answering Systems: 1969, Communications of the ACM, V. 13, No.1, pp. 15-30, January 1970.

4. Heidorn, G.E., Natural Language Inputs to a Simulation Programming System, Research Proposal submitted to the Office of Naval Research, October 1970.

5. Hansen, R.C., GES: A Data Structure-to-GPSS Encoding System, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1970.

6. McGee, R.T., The Translation of Data Structure Representations of Simple Queuing Problems into GPSS Programs and English Text, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1971.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst Professor G.E. Heidorn, Code 55 Hd Department of Operation Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	5
4.	Asst Professor G.H. Syms, Code 53 Zz Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
5.	LCDR E.S. Baker, USN 2372 Murray Ridge Road San Diego, California 92123	2

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	Unclassified
	2b. GROUP

3 REPORT TITLE

Question-Answer Inputs to a Simulation Program Generating System

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Master's Thesis; June 1971

5 AUTHOR(S) *(First name, middle initial, last name)*

Eldon S. Baker

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1971	99	6

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

One of the objectives of a research effort at the Naval Postgraduate School is the realization of a "simulation program generating system" to produce executable computer programs from natural language descriptions of simulation problems. The basic part of the system being developed is a FORTRAN program for translating input text into a representative data structure and constructing output text from the data structure. Previous work has resulted in the capability to translate the data structure for a simulation problem into an English text description and a GPSS computer program. The purpose of this thesis was to supplement the above with an interactive question-answer scheme that generates the simulation problem data structure. The resulting GPSS "simulation program generating system" is capable of handling a variety of simple queuing problems.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31409

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Simulation						
Program Generator						
Question-Answer						
Natural Language Processing						

Thesis 142001
B1685 Baker
c.1 Question-answer inputs
to a simulation program
generating system.